FREIE UNIVERSITÄT BERLIN

Swarm Approaches For Semantic Triple

Clustering And Retrieval In

Distributed RDF Spaces

Sebastian Koske, M.Sc.

B-09-04
February 2009

FACHBEREICH MATHEMATIK UND INFORMATIK
SERIE B • INFORMATIK

Abstract

Within the last decade, many new approaches have been developed towards a shift from the "classic" web, which is designed primarily for human use, to a machine-processable web, which can be used by heterogenous systems to cooperate in an independent and scalable manner. These approaches include new technologies for modeling semantic data to be commonly understood by applications such as data miners, web service clients, autonomous software agents, or reasoning tools, as well as new coordination models for a loose and scalable coupling of independent systems, varying from high-end servers to small embedded applications in PDAs, cell phones, or sensor nets.

Especially the combination of semantically modeled domain knowledge and common web services to become *semantic web services* seems to be a promising technology in the respect of an internet-scale integration model. They overcome the tightly coupled message exchange pattern which is used in classic Web Services, but use *Tuple Spaces* instead, which accompany the Word Wide Web's core paradigm of information exchange via persistent publication [1].

As those technologies imply internet-scalability, it is essential to investigate how decentralized and fully distributed architectures can be realized without significant performance impacts. Observing natural societies like swarms, flocks, or hives, they seem to provide many of the desired characteristics, such as scalability, dynamism, failure tolerance and simplicity.

Swarm strategies are already successfully used in the field of peer-to-peer networking and special cases of linear optimization. This thesis implements and evaluates swarm-based approaches for semantic Tuple Spaces, based on the previous work of Daniel Graff, who implemented a LINDA Tuple Space system called SwarmLinda [2]. It extends this space with semantic triple clustering and retrieval by adapting common similarity measures for a distributed swarm-based architecture and by developing research strategies inspired by ant routing algorithms.

Contents

Contents **5**

1 Introduction **11**

1.1 Scientific Context 11

1.2 Research Objectives 12

1.3 Outline 13

2 State of Research **15**

2.1 Semantic Modeling Languages 15

2.1.1 RDF 15

2.1.2 RDF-S 16

Class Definement 16

Property Definement 17

Predefined Classes and Properties 17

2.1.3 OWL 18

2.2 Space-Based Coordination 19

2.2.1 LINDA 19

Reference Independency 19

Time Independency 20

Space Independency 20

Primitives 20

Insufficiencies 21

2.2.2 Triple Space Computing 21

2.3 Semantic Similarity and Relatedness Measures 22

2.3.1 Distance-Based Measures . . . 23
Rada, Mili, Bicknell, and Blettner . . . 24
Castano, Ferrara, Montanelli, and Racca . . . 25
Haase, Siebers, and Harmelen . . . 25
Leacock and Chodorow . . . 26
Wu and Palmer . . . 26
Hirst and St-Onge . . . 27
2.3.2 Information-Theoretic Measures . . . 28
Lin . . . 29
Resnick . . . 31
2.3.3 Comparison . . . 31
2.4 Swarm Intelligence . . . 32
2.5 SwarmLinda . . . 33
2.5.1 NetLogo . . . 33
2.5.2 SwarmLinda Extensions . . . 34
OUT - Ants . . . 34
IN / RD - Ants . . . 35
Scout - Ants . . . 35
Conclusions . . . 35
2.6 Related Work . . . 36
2.6.1 JavaSpaces . . . 36
2.6.2 GigaSpaces . . . 37
2.6.3 TSpaces . . . 37
2.6.4 XMLSpaces.NET . . . 37
2.6.5 eLinda . . . 38
2.6.6 Tripcom Project . . . 38

3 Approach **39**
3.1 Previous Works and Objectives . . . 39
3.2 Overview . . . 40
3.2.1 A-Box Clustering . . . 40
3.2.2 T-Box Clustering . . . 41

3.2.3 Triple Retrieval . . . 42
3.2.4 Triple Space Architecture . . . 43
3.3 Type Identification . . . 44
3.4 Similarity Measure Evaluation . . . 45
3.4.1 Distance-Based Measures . . . 46
3.4.2 Information-Theoretic Measures . . . 49
3.4.3 Conclusion . . . 49
3.4.4 Measure Modifications . . . 50
Degree Normalization . . . 50
Measure Scaling . . . 51
3.5 Notation . . . 54
3.6 A-Box Clustering . . . 55
3.6.1 Similarity Distribution . . . 55
3.6.2 OUT-Ants . . . 55
3.6.3 Initialization . . . 56
3.6.4 Learning . . . 56
3.6.5 Task Fulfillment . . . 56
3.6.6 Path Selection . . . 57
3.6.7 Transition . . . 58
3.6.8 Returning . . . 59
Taken Path Strategy . . . 59
Shortest Path Strategy . . . 60
3.6.9 Completion and Death . . . 60
3.7 S-Box Clustering . . . 60
3.8 Triple Retrieval . . . 61
3.8.1 Concurrent Matching . . . 61
3.8.2 Task Fulfillment . . . 62
3.8.3 Distributed Locking . . . 62
3.8.4 Completion . . . 63
3.9 Node Processes . . . 64
3.9.1 Triple and Template Processing . . . 64

3.9.2 Cluster Maintenance . . . 64
3.9.3 Pheromone Decrease . . . 65
3.9.4 Template Generation . . . 65
3.9.5 Garbage Collection . . . 65

4 Implementation **66**
4.1 Installation . . . 66
4.1.1 Requirements . . . 66
Java . . . 66
Eclipse . . . 66
Ant . . . 67
Maven . . . 67
4.1.2 Software Overview . . . 67
Projects . . . 67
Libraries . . . 69
4.1.3 Simulator Build and Launch . . . 69
4.2 User Manual . . . 70
4.2.1 Network Creation . . . 70
4.2.2 Simulator . . . 71
Main View . . . 71
Layout Settings . . . 71
URI-Cluster Configuration . . . 71
Semantic Cluster Configuration . . . 73
Input Area . . . 74
Cluster Control . . . 75
Monitoring . . . 75
Evaluation . . . 76
4.3 Implementation Notes . . . 78
4.3.1 Concurrency in NetLogo . . . 78
4.3.2 Concurrency in the Extensions . . . 78
4.3.3 Main Data Structures . . . 78
ABox . . . 78

S-Box . . . 79
Triple Distributions and Scent-Lists . . . 83
4.3.4 Conclusion . . . 83

5 Evaluation **85**
5.1 Clustering Evaluation . . . 85
5.1.1 Entropy Calculation . . . 85
Semantic Entropy . . . 85
Spatial Sematic Entropy . . . 87
Spatial Sematic Entropy Gain . . . 87
5.1.2 Test Scenarios . . . 88
5.1.3 Entropy Evaluation . . . 89
5.1.4 Local Similarity Gain . . . 92
5.1.5 Semantic Neighborhoods . . . 92
5.1.6 S-Box Clustering . . . 96
5.1.7 Additional Configuration Options . . . 99
Minimum Similarity Bound . . . 99
Time-To-Live . . . 100
Maximum Ant Count . . . 100
Pheromone Decay Rate . . . 101
5.2 Retrieval Evaluation . . . 102
5.2.1 Cluster Processes and Semantic Trail Quality . . . 102
5.2.2 Search Path Evaluation . . . 104

6 Conclusions and Future Work **106**
6.1 Conclusions . . . 106
6.1.1 Accomplishments . . . 106
6.1.2 Observations . . . 107
6.1.3 Limitations . . . 108
6.2 Future Work . . . 109
6.2.1 Maintenance Improvements . . . 109
6.2.2 Correlation-Based Decisions . . . 110

6.2.3 Distributed Shortest Path 110
6.2.4 Further Suggestions 111

List of Figures **112**

List of Listings **114**

List of Definitions **115**

List of Acronyms **116**

Index **119**

Bibliography **123**

A Further Evaluation Results **129**

B Legal Notices **141**

Chapter 1

Introduction

1.1 Scientific Context

"The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. [...] In the near future, these developments will usher in significant new functionality as machines become much better able to process and understand *the data that they merely display at present" [3].*

In today's World Wide Web (WWW), resources such as web pages and their relations are analyzed mainly by their internal structure. Word frequencies provide information about the content of a page, headers and links provide information about its context. This approach lacks a common technique which allows content and context to be expressed and processed in a semantic way. Yet, such a facility is necessary to exchange semantic information seamlessly between heterogenous services, applications and enterprises. Furthermore, software agents could use semantic data and services in order to perform highly integrative tasks through a variety of systems across the web.

The Semantic Web Community [4, 5] assembles different technologies like the Resource Description Framework (RDF), the SPARQL Protocol and RDF Query Language (SPARQL) or the Web Ontology Language (OWL) in order to establish semantics in the web. These technologies allow the modeling of domain knowledge and taxonomies by describing concepts (classes) of resources, their properties, restrictions and inter-concept relations by so called *web ontologies*. Such an ontology may also include rules that afford artificial intelligence engines to conduct reasoning and to perform automated knowledge interference within a given domain. The World Wide Web Consortium (W3C) recommends the RDF as a metadata facility for describing web resources and their relations, using machine-processable triples of the form *(subject, predicate, object)*. Thus, the RDF specifications provide a lightweight framework that can be used to describe ontologies for various application scenarios. As the set of triples needed for a complete

domain description may become very vast, it raises the issue of efficient storage and retrieval.

In his work "Generative communication in Linda" (1985) [6], David Gelernter from Yale University introduced a coordination language for multi processor computation called *LINDA*. He suggested that process coordination and data exchange could be established using a virtual associative memory - a *Tuple Space*, where data, represented as n-Tuples, can be accessed via simple primitives like OUT, IN and RD. These tuples contain values of primitive types (like numbers or strings) and are retrieved using templates that contain values, value types or wildcards.

Daniel Graff combined this architecture with swarm-based strategies to create a distributed LINDA Tuple Space [2]. In his approach, the tuple clustering and retrieval is achieved by a swarm of virtual ants using special pheromones for each value type. He implemented this system using NetLogo, a Java-written network simulator. By evaluating the spatial triple entropy, he showed that swarm approaches are capable of dynamically congregating triples by kind and can increase the level of order in the triple set. Concurrently to this thesis, the bachelor thesis of Anne Augustin investigates a clustering of RDF-triples, which bases on the Uniform Resource Identifier (URI) of resources [7].

1.2 Research Objectives

The primary motivation of this thesis is to extend the implementation of Daniel Graff to store RDF ontologies. Instead of n-tuples, which contain values of pre-defined primitive types, RDF-triples contain resources of semantically interlinked classes and properties. Therefore, a clustering of RDF-triples must regard the underlying ontological structure and consider the semantic similarity of their resources.

Daniel Graff's work is extended in the bachelor thesis of Anne Augustin, who investigates the possibility of clustering RDF-triples by resource URI [7]. This approach bases on the assumption that resources with similar URIs are also semantically related. Although this assumption may be completely valid in some specialized ontologies, in general it must be considered merely as a heuristic. Moreover, since basic ontologies, having specific namespaces, are often shared for interoperability, the inversion of the assumption is generally not true. If, for instance the following triples are considered,

(http://inf.fu-berlin.de#tolk,rdf:type,http://general#lecturer),
(http://physik.fu-berlin.de#brewer,rdf:type,http://general#lecturer),
(http://inf.fu-berlin.de#cafeteria,rdf:type,http://general#refectory)

the lecturer instances should be considered more similar due to their common class.

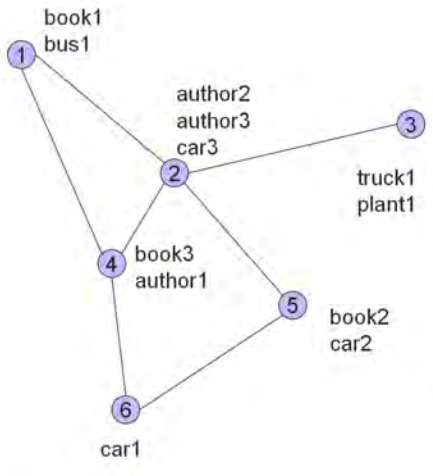


Figure 1.1: Spatial triple distribution in SwarmLinda URI-Extensions.

As a result, URI-clustering fails to identify related resources with different identifiers and may result in a very poor semantic clustering. Figure 1.1 illustrates how triples, corresponding to resources of the same type, may spread all over the network using URI-clustering.

Consequentially, triples clustered by URI cannot efficiently be retrieved by type – yet tuple retrieval via typed templates is not only part of the original LINDA model, but is also essential for semantic services and must therefore be considered to be indispensable.

In order to provide support for a typed triple retrieval (via typed templates), it becomes inevitable to group statements about similar resources together, to form semantic neighborhoods within the space (see Figure 1.2). Therefor this thesis investigates the possibility of distributing and utilizing the very contents of the ontologies to dynamically infer the semantic similarity of resources. It introduces, implements, and evaluates adaptive swarm-based approaches for creating and maintaining semantic neighborhoods, allowing a scalable and yet efficient typed triple retrieval.

1.3 Outline

This thesis is divided into six chapters. While the first chapter consists of this introduction, the second chapter gives a detailed overview of the current state of research. It explains the basic features of RDF and RDF-Schema (RDF-S) and describes the LINDA coordination model. It also provides an overview of Triple Space Computing and describes the efforts towards semantic web services. As semantic clustering requires a quantification of similarity, common similarity measures are discussed and compared towards their suitability for a possible usage in a distributed system. Furthermore, this chapter examines applied swarm intelligence in the field of linear optimization and describes how it is used to find fair solutions for the Traveling Salesperson Problem (TSP). It also

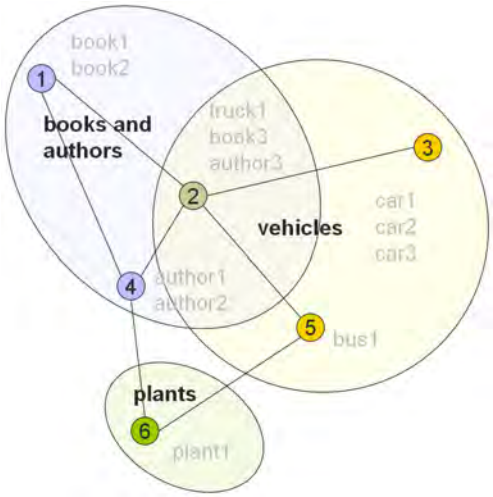


Figure 1.2: Spatial triple distribution in Semantic SwarmLinda.

contains comprehensive surveys of the previous works of Daniel Graff and Anne Augustin as well as further related works in the field of clustered Spaces.

The third chapter provides detailed explanations of the semantic clustering and retrieval approaches. It discusses the necessity to distinguish between the different layers of RDF and introduces appropriate strategies. Furthermore, the overall cluster architecture is explained, including all participating processes and underlying calculations.

The system implementation is described in chapter 4. It includes a manual of the simulator interface, explains its functionality and addresses the most crucial implementation details.

Chapter 5 covers the system evaluation. For this purpose, appropriate quality indicators of the semantic clustering and retrieval are introduced. These indicators are applied in a variety of tests conducted on the different aspects of the system. Their results are presented and discussed in detail in this chapter as well.

The sixth chapter presents the final conclusions of this thesis and contains suggestions for possible future works.

Chapter 2

State of Research

This chapter provides the scientific expertise which is used in this thesis. It describes the semantic modeling languages RDF, RDF-S and OWL and introduces the LINDA coordination model. Furthermore, it contains an introduction of common similarity measures, in which their characteristics and features are analyzed and compared. It also describes the efforts of Daniel Graff and Anne Augustin, which are extended and utilized in this work, and gives an overview existing Triple Space implementations.

2.1 Semantic Modeling Languages

2.1.1 RDF

Regarding its specification, the RDF integrates a "variety of applications from library catalogs and world-wide directories to syndication and aggregation of news, software, and content to personal collections of music, photos, and events using XML as an interchange syntax" [8]. RDF expressions are formed by a set of triples, each consisting of a subject, a predicate and an object. These triples define an RDF graph, in which classes, instances of such and values are represented by nodes and properties are represented by directed arcs [9]. Figure 2.1 shows how domain knowledge can be represented by such a graph.

Within RDF, resources are either identified by a URI with an optional fragment identifier, a literal value, or blank. In standard RDF, all literals are *XML Literals*, thus typed literals are expressible via XML-Schema fragments (e.g. *<xsd:boolean,"true">*) [11–13]. Although not enforced, it is common to use URI-schemes that start with a specific *namespace*. The RDF recommendation uses special namespaces for its set of standard vocabulary, which is described next.

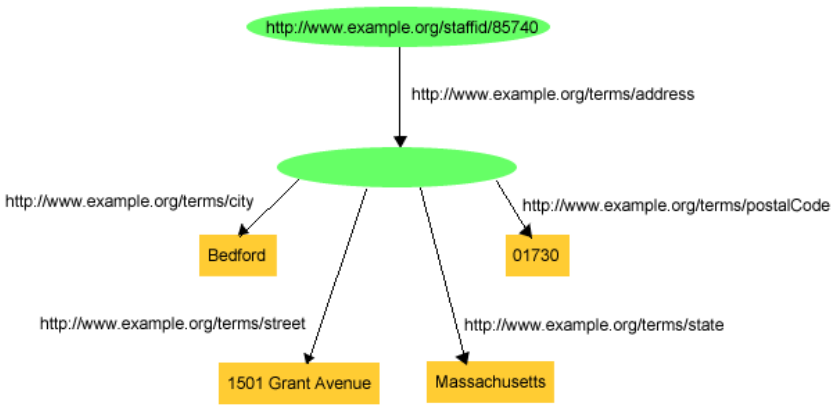


Figure 2.1: RDF graph, derived from S-P-O triples [10]

2.1.2 RDF-S

The predefined RDF-S vocabulary is used to express domain ontologies by class hierarchies, properties, class instances (individuals) and values. In the following sections, the namespace prefix *rdf* is used for the the official RDF namespace[1] and *rdfs* is used as a substitute for the RDF-S namespace[2], as suggested by the W3C. The next paragraphs introduce the most important constructs [14].

Class Definement

rdfs:Class A standard resource that represents the general *class* construct. It is the base class of all resources that are RDF-classes and is an instance of *rdfs:Class* itself.

rdfs:Resource A standard resource that represents the top level class of all resources, hence all resources are instances of *rdfs:Resource*. It is also an instance of *rdfs:Class*.

rdfs:Literal This subclass of *rdfs:Resource* states the base class of all literal values, plain such as strings and integers, or typed (custom or via XML-Schema). Hence, all literal values are instances of *rdfs:Literal*. Typed literals are instances of the *rdfs:Datatype* class, yet RDF-S does not define a special class of plain literals.

rdfs:Datatype is the top-level class of data types and a subclass of *rdfs:Literal*.

rdf:XMLLiteral is the base class of XML literal values. It is an instance of *rdfs:Datatype* and a subclass of *rdfs:Literal*.

rdf:Property represents the base class of all RDF properties.

[1] http://www.w3.org/1999/02/22-rdf-syntax-ns#
[2] http://www.w3.org/2000/01/rdf-schema#

Property Definement

In RDF, properties are defined using the following instances of *rdf:Property*.

rdfs:range *(P, rdfs:range, C)* states that all values of property P are instances of class C. Hence, in any triple of the form *(?, P, X)*, X is an instance of C. If more than one definition of the kind *(P, rdfs:range, C_i)* is given for P, the values of P must be an instance of all classes $C_1 \dots C_n$.

rdfs:domain *(P, rdfs:domain, C)* states that any resource which has a property P, is an instance of class C. Hence, in any triple of the form *(X, P, ?)*, X is an instance of C. If more than one definition of kind *(P, rdfs:domain, C_i)* is given for P, any resource with property P must be an instance of all classes $C_1 \dots C_n$.

rdf:type *(R, rdf:type, C)* states that the resource R is an instance of class C.

rdfs:subClassOf *(A, rdfs:subClassOf, B)* states that all the instances of class A are also instances of class B, letting A become a subclass of B.

rdfs:subPropertyOf states that all resources related by one property are also related by another. Given the triples *(P_2, rdfs:subPropertyOf, P_1)* and *(R_1, P_2, R_2)*, it means that resource R_1 is related to R_2 by P_2 and therefore also by P_1.

rdfs:label may be used to provide a human-readable name of a resource.

rdfs:comment may be used to provide a human-readable description of a resource.

Predefined Classes and Properties

RDF-S provides a set of predefined classes for general purpose use. Among them, there are four container classes to define different kinds of collections. A collection is defined by introducing an instance of a container class and by a set of RDF-triples defining its elements. The containers provided are [10, 14]:

rdfs:Container is the super-class of all container classes.

rdf:Bag A *bag* represents a group of resources or literals, possibly including duplicate members, where there is no significance in the order of the members.

rdf:Seq A *sequence* represents a group of resources or literals, possibly including duplicate members, where the member order is significant.

rdf:Alt An *alternative* represents a group of resources or literals that are used alternatively (typically used for a single value of a property). For example, an *alternative* might be used to describe alternative language translations for the title of a book, or to describe a list of alternative internet sites at which a resource might be found.

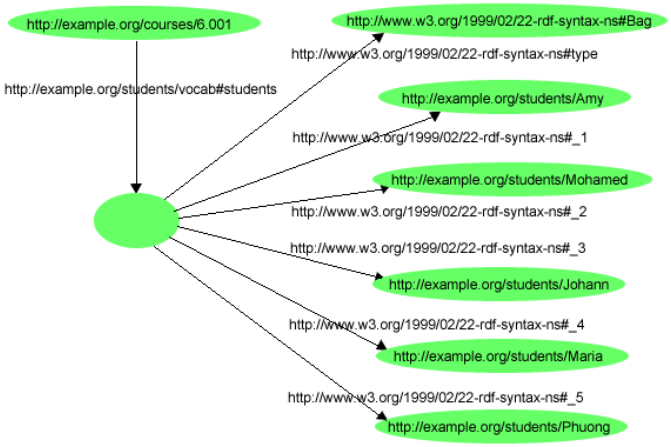


Figure 2.2: RDF container membership represented by S-P-O triples [10]

Container membership can be declared using special pre-defined instances of *rdfs:ContainerMembershipProperty*, which are sub-properties of *rdfs:member* and have names of the form *rdf:_n*. There, n is a decimal integer greater than zero, e.g. *rdf:_1*, *rdf:_2*, and so on [10]. Figure 2.2 gives an example of a possible container membership definition. Yet, as those membership definitions are open and unbound, there is no way to state the final element of a container. For this purpose, RDF-S provides linked lists which have a common *head* and *tail* structure and are terminated by the *rdfs:nil* resource. Figure 2.3 gives an example of a possible list definition. The constructs provided for list-like constructions are [10, 14]

rdf:List is an instance of *rdfs:Class* and the base class of all lists and list-like structures.

rdf:first is an instance of *rdf:Property* and defines the *head* element of a list.

rdf:rest is also an instance of *rdf:Property* and defines the *tail* of a list.

rdf:nil is an instance of *rdf:List* and represents an empty list or list-like structure.

2.1.3 OWL

The Web Ontology Language (OWL) is a family of languages based on RDF and RDF-S. It "has more facilities for expressing meaning and semantics than XML, RDF and RDF-S and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web" [15]. OWL provides the following three levels of expressiveness [15–18].

OWL Lite is the least complex sub-language. It provides mechanisms for defining property cardinalities (0 or 1 only) and is designed for modeling simple taxonomies and to allow basic inference.

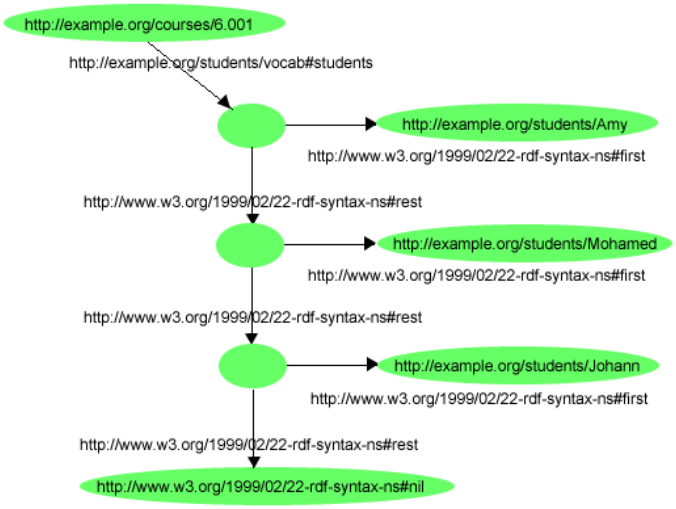


Figure 2.3: RDF List definition represented by S-P-O triples [10]

OWL DL provides the maximum amount of expressiveness that guarantees all conclusions to be decidable and computable in finite time. Therefore, it allows the usage of the same language constructs as *OWL Full*, but under certain limitations. For example, classes cannot be instances of other classes and properties cannot be individuals or classes. Due to its completeness and expressiveness *OWL DL* is the most commonly used sub-language of OWL.

OWL Full allows the full usage of RDF-S without restrictions. Expressions in *OWL Full* may me contradictions and conclusions may not be computable.

2.2 Space-Based Coordination

2.2.1 LINDA

LINDA was introduced by David Gelernter as a loosely coupled process coordination model [6]. He suggested that data exchange could be accomplished through an associative memory (Tuple Space), where processes could put and retrieve n-tuples asynchronously via pre-defined primitives, as shown in figure 2.4. Though very simple, this model provides some profound advantages related to parallel process decoupling and building reusable, distributed, heterogenous and agile communicating applications [1, 19]. These advantage are explained in the following.

Reference Independency

Coordinated by just the Tuple Space as a middleware, processes do not have to directly know each other. Tuple production and consumption are completely independent and it

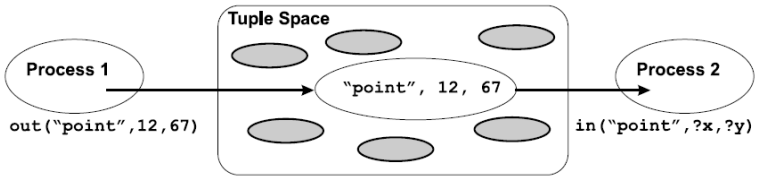


Figure 2.4: A simple one-to-one communication pattern [20]

is not required for participating processes to explicitly register or synchronize with each other, since they only communicate via tuple exchange.

Time Independency

As the Tuple Space guarantees persistent data storage, process communication can be performed in a complete asynchronous manner. Tuple producers and consumers may access the space at any time and in any order. The only time dependency is, of course, that a triple has to be published before it can be retrieved.

Space Independency

As long as different processes have access to the same Tuple Space, they can run in completely different environments of hardware or operating systems.

Primitives

To access the space, Gelernter proclaimed the following LINDA primitives.

OUT A tuple is stored in the space using the primitive $out(c_1, \ldots, c_n)$. As a result, the tuple $(c_1, \ldots, c_n)$ becomes available for all other processes.

IN Tuples are retrieved using the IN-primitive with *tuple templates* (or *anti-tuples*) as parameter. A template may contain values, types or wildcard variables. A tuple t matches a template $\bar{t}$ if they have the same number of components and each component c_i matches the component $\bar{c}_i$ of the template. A component c_i matches $\bar{c}_i$ if $\bar{c}_i$ is a wildcard variable (in which case the variable is bound to c_i), if $\bar{c}_i$ is the type of c_i, or if $\bar{c}_i = c_i$. This primitive blocks the requesting process until a matching tuple is found. It also removes the found tuple from the space. If more than one match exists, only one of them is returned.

INP The non-blocking version of the IN-primitive, indicating a failure if no tuple was found.

RD The RD-primitive works like the IN-primitive, but without removing the found tuple. In this respect is a non-destructive version of the IN-primitive.

RDP The non-blocking version of the RD-primitive, that also indicates a failure if no tuple was found.

EVAL This primitive is used to create new processes that evaluate triples and write the results back to the space.

Insufficiencies

The LINDA coordination model has three major insufficiencies [20–22]. The first shortcoming is that the paradigm allows tuples to only hold primitive values. The types of these values have to be commonly agreed on by all participating processes, since there is no facility to dynamically state the semantics of tuples (data scheme dependency). Furthermore, the LINDA model lacks a sophisticated tuple selection facility, as matching is conducted only by the value or by the type of the tuple components. In later implementations, see section 2.6 (Related Work), page 36, tuples are also allowed to hold complex objects and custom matchings are supported as well. The third insufficiency is scalability. Originally, there was only one global space proposed, foiling a distributed architecture. In later works, clusters of Tuple Space have been suggested to cooperate in tuple hosting.

2.2.2 Triple Space Computing

Given the described technologies of semantic modeling languages like RDF and OWL, and a loosely coupled coordination model such as Tuple Spaces, the next step towards a semantic web is the introduction of semantic services for the web – called *semantic web services*. Although the name may imply that this addresses only semantic extensions to common web services, is goes beyond just that as it is also about a change of the current paradigm of web services [1, 19, 22].

Although sharing the term *web*, the WWW and web services currently differ in their very basic principles. While the WWW follows the paradigm of Representational State Transfer (REST), which decouples content provider and consumer, web services are basically designed for data exchange by tightly coupled SOAP-message exchange, generally using synchronous HTTP transactions. Since the WWW gains its vast scalability from following the REST principle, it seems to be compulsory for services targeting a sematic web to also coordinate via Representational State Transfer. Hence, the combination of space-based coordination technologies and semantic modeling facilities is a promising approach towards real semantic web services (see figure 2.5). Those efforts have been increased within the last years and form a new field within the area of distributed computing called *Triple Space Computing*. With semantic extensions, tuple spaces might also overcome data scheme dependency, since descriptions of tuples types and contents can be processed and understood dynamically by all clients. Furthermore, sophisticated semantic-aware matching rules could be defined to find tuples not only

by their structure, but by using high-level semantic queries expressed in an RDF query language such as SPARQL or the RDF Data Query Language (RDQL) [1].

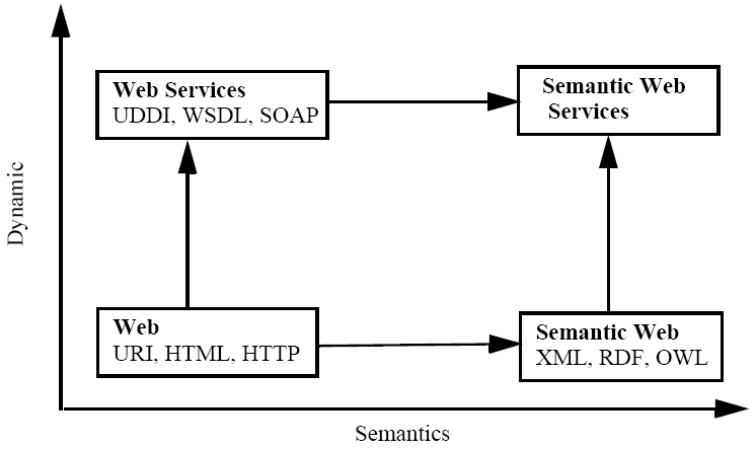


Figure 2.5: The four major stages in the development of semantic web services [19]

2.3 Semantic Similarity and Relatedness Measures

Before sematic modeling languages were introduced, similarity measures were mostly restricted to context-free, non-semantic word comparison. Hence, these measures basically addressed string similarity, though there were also other approaches like phonetic similarity analysis. For a better understanding of the limitations of these approaches, the most commonly used measures for string similarity are briefly described in the following.

Longest Common Subsequence This measure compares two or more strings s_i by the longest character sequence they have in common. This sequence is typically identified using the Z-Algorithm or the algorithm of Boyer-Moore. The longer that sequence is, the more similar the strings are assumed to be.

Levenshtein Distance This is a measure for the shortest editing distance between two strings. An editing step can be an insertion of a gap, a character alteration or a character deletion. Depending on the editing step penalties defined, the length of the shortest editing distance (Levenshtein distance) between two strings defines their similarity.

Vector-Based Measures consider strings as vectors of characters. Using a certain character distance function d, it is possible to use the vector distance (for instance the euclidic distance) or the vector angle cosine as a similarity measure.

Pattern Matching There are numerous methods of finding patterns and matches for regular expressions in strings. Most of them make use of data structures like

Suffix Trees or *Suffix Arrays*. In these approaches, strings are considered similar if they include similar patterns. This method is, for example, applied to a large extend in the field of gene sequence comparisons.

The problem is of course, that real-world concepts are generally represented by multiple strings. They are expressed in different languages and in many cases by sets of synonyms. As concept similarity is also almost never reflected by word similarity, the described approaches generally fail to derive any semantic context. This necessitates to establish completely different strategies to define semantic similarity in web ontologies, which utilize the very modeled information.

For the purpose of discussing some of these semantic similarity measures in the next sections, the general needs for a similarity function *sim* must de defined first. Let C be the set of concepts of a given ontology O, we define

$$\begin{gathered} sim : C \times C \mapsto [0 \dots max] \\ \forall c \in C : sim(c, c) = max \\ \forall c_1, c_2 \in C : sim(c_1, c_2) = sim(c_2, c_1) \end{gathered}$$

where max is a measure-specific value that states the maximum similarity of two concepts $c_1, c_2 \in C$. In order to enable the combination and comparison of different measures, they should be normalized that way $max = 1$.

2.3.1 Distance-Based Measures

An ontology O defines a set of classes C and a set of interlinking properties P, which can be translated to a directed graph. As illustrated in figure 2.6, these graphs are divided into different levels of abstraction. The *RDF-S-Level* contains all classes and properties pre-defined in the RDF specification. The domain-specific classes and properties define the *Schema-Level*. The unification of both levels is also called the *T-Box* of O. The *Data-Level* consists of class instances (individuals) and values and is also called the *A-Box* of O.

Semantic similarity of concepts is formed by their relationships, which are expressed via properties. Regarding the different abstraction layers, not all properties are suitable for semantic considerations in any case. Individuals either refer to one or more classes via the *rdf:type* property, or they implicitly belong to *rdfs:Resource*. Thus, in this case, the *rdf:type* property is similarity relevant. In the T-Box, *rdf:type* is used to define classes and properties. Since the simple fact that two resources are classes or properties does not imply semantic similarity, *rdf:type* is not similarity relevant in this abstraction level.

Most common graph distance-based approaches are restricted to the *rdfs:subClassOf* or *rdfs:subPropertyOf* hierarchy among resources. They argue that RDF types (classes and properties) are similar if their least common type within the ontology is close to both of them. This restriction has certain limitations. Although the similarity detected between concepts like *apple* and *orange* is intuitively appropriate, since they are both a *fruit*, it

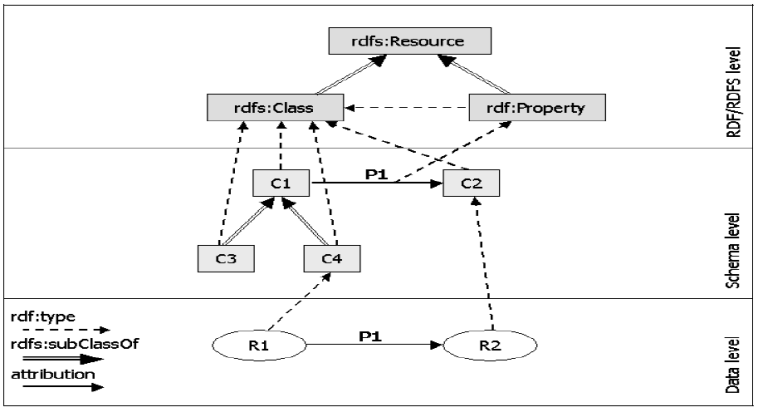


Figure 2.6: Abstraction levels in a typical RDF-S ontology [23, page 3]

does exclude similarity formed by other describing properties that represent relations like *has part*, *part of*, *contains* or *uses*. Though one can argue that *oil* and *car* are not similar concepts, they are certainly related. If the observed concept is *motor*, this relatedness becomes comprehensible.

In the next sections, some of the most important distance-based algorithms are discussed. Unless stated otherwise, they are restricted only to the type hierarchy of resources [24].

Rada, Mili, Bicknell, and Blettner

In [25] Rada et al. introduce a basic distance measure D_s between two concepts c_1 and c_2. They use the length of the shortest path within the ontology graph between c_1 and c_2 (*edge counting*) as similarity indicator. If such path does not exist, D_s is 0. Since the similarity is anti-proportional to the graph distance, it is defined as

$$sim_{Rada}(c_1, c_2) = \frac{1}{1 + D_s(c_1, c_2)}$$

Adding 1 to the length of the shortest path prevents an infinite self-similarity of concepts, as D_s is zero in this case.

The disadvantage of this measure is that it is unaware of the root distance of concepts. In an exemplary ontology, this measure states the same similarity between the two assumed top-level classes *Thing* and *Living Being* as between a *Self-Sealing Stem Bolt* and a *Conventional Stem Bolt*, both being *Stem Bolts* and located much deeper in the ontology. Since deeper concepts are generally more specific, a larger root distance of the least common subsumer (LCS) of two concepts should be reflected by an increased similarity.

Castano, Ferrara, Montanelli, and Racca

Castano et al. extend the measure of Rada et al. by considering not only *is-a* relations [26]. They call their measure *affinity* and let affinity weights ranging from 0 to 1 be defined for any property. The *total affinity* of a path is computed by multiplying the weights of its edges. Hence, their measure is

$$sim_{Cast}(c_1, c_2) = \begin{cases} \max_{i=1\ldots k}(W_{t \mapsto {}^{n}_{i} c_2}) & \text{if } k \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

There, k is the number of paths between c_1 and c_2, and $t \mapsto {}^{n}_{i}c_2$ denotes the ith path of length $n \geq 1$. $W_{t \mapsto {}^{n}_{i} c_2} = \prod_{j=1}^{n} w_{i,j}$, where $w_{i,j} \geq 0$ is the affinity weight of the jth property in path i. Though not explicitly stated, it is assumed, that a path from c to itself has a length and affinity weight of 1. Their measure is normalized, as all multiplied weights are within range $[0 \ldots 1]$ and therefore $max_{sim_{Cast}} = 1$.

Haase, Siebers, and Harmelen

Haase, Siebers, and Harmelen take the concept depth into account and suggest the following similarity measure, where l is the length of the shortest path between c_1 and c_2 and h denotes the height of the LCS of c_1 and c_2 [27].

$$sim_{Haas}(c_1, c_2) = \begin{cases} e^{-\alpha l} \cdot \dfrac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} & \text{if } c_1 \neq c_2 \\ 1 & \text{otherwise} \end{cases}$$

The measure provides the two scaling variables $\alpha, \beta \geq 0$ to adjust the influence of the concept distance (α) and the LCS height (β) independently. Based on their benchmarks[3], they find $\alpha = 0.2$ and $\beta = 0.6$ to be the optimal values [28, 29]. The maximum similarity of Haase-Siebers-Harmelen is 1 as shown (2.3), therefore the measure is normalized.

$$\forall \alpha, l \geq 0 : 1 \leq e^{\alpha l} \quad \Rightarrow \quad 0 \leq e^{-\alpha l} \leq 1 \tag{2.1}$$

$$\forall \beta, h \geq 0 : e^{\beta h} - e^{-\beta h} \leq e^{\beta h} + e^{-\beta h} \quad \Rightarrow \quad 0 \leq \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} \leq 1 \tag{2.2}$$

$$(2.1) \wedge (2.2) \quad \Rightarrow \quad sim_{Haas} \mapsto [0 \ldots 1] \tag{2.3}$$

[3] They used the measure for an evaluation of the *Association for Computing Machinery, Inc. (ACM) Computing Classification System*.

Leacock and Chodorow

Leacock and Chodorow also define a measure that is based on the length of the shortest path between concepts [30]. In contrast to Haase, Siebers, and Harmelen, they use a logarithmic scale to flatten similarity decrease caused by increased distance.

$$sim_{LC}(c_1, c_2) = -\log \frac{length(c_1, c_2)}{2D}$$

In this definition $length(c_1, c_2)$ is the number of distinct nodes on the shortest path between c_1 and c_2 and D is the maximum depth of the ontology. Although the upper bound of this measure is constant within a given ontology, it is not 1. Since the maximum similarity occurs between a concept and itself, it is

$$max_{sim_{LC}} = -\log \frac{1}{2D} = \log(2D).$$

Hence, a normalized version of sim_{LC} can be constructed using $max_{sim_{LC}}$ as scaling factor.

$$\begin{aligned} sim^*_{LC} &= \frac{sim_{LC}}{max_{sim_{LC}}} = \frac{-\log \frac{length(c_1, c_2)}{2D}}{\log(2D)} \\ &= \frac{\log(2D) - log(length(c_1, c_2))}{\log(2D)} = 1 - \frac{\log(length(c_1, c_2))}{\log(2D)} \\ &= 1 - \log_{2D} length(c_1, c_2) \end{aligned}$$

Wu and Palmer

Wu and Palmer also face the issue of normalization in [31]. Their measure is based on the depth of a concept, which is the number of nodes on the path to the root (see figure 2.7).

$$sim_{WP}(c_1, c_2) = \frac{2 \cdot depth(lcs(c_1, c_2))}{depth(c_1) + depth(c_2) + 2 \cdot depth(lcs(c_1, c_2))}$$

In [32] Resnik reformulates this measure to

$$sim^R_{WP}(c_1, c_2) = \frac{2 \cdot depth(lcs(c_1, c_2))}{depth(c_1) + depth(c_2))}$$

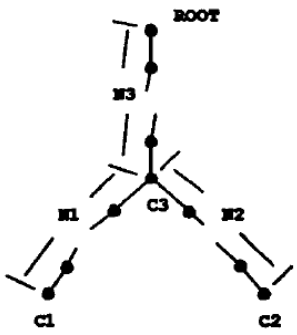


Figure 2.7: The conceptional similarity measure of Wu and Palmer (example) [31, page 136]. The similarity between $C1$ and $C2$ is $sim_{WP}(C1, C2) = \frac{2 \cdot N3}{N1 + N2 + 2 \cdot N3}$

Hirst and St-Onge

Hirst and St-Onge suggest a measure which considers any property [33]. They classify property relations by their direction: *horizontal*, *upwards* and *downwards*.

Upward Relations are generalizations such as between *apple* and *fruit* (via *is-a*).

Downward Relations are specializations and the inverse to upward relations.

Horizontal Relations other relations that express relatedness are considered *horizontal* (e.g. *successor*, *predecessor*, *causes-effect*).

Moreover, Hirst and St-Onge define that legal paths must only contain the allowed changes of directions (see figure 2.8), which are

R1 No other direction may precede an upward relation – once a link that narrows down the context (downward or horizontal) has been used, it is not permitted to enlarge the context again by using an upward link.

R2 At most one change of direction is allowed, except that it is permitted to use a horizontal link to make a transition from an upward to a downward direction.

Using the constants C and $k \geq 0$, the similarity of two concepts is defined as

$$sim_{HO} = \frac{1}{C - \text{path length} + k \cdot \text{number of changes of direction}}$$

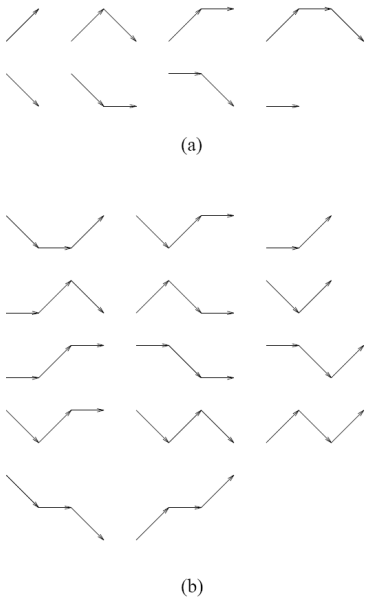


Figure 2.8: (a) Patterns of paths allowable medium-strong relations and (b) patterns of paths not allowable. (Each vector denotes one or more links in the same direction.) [33, page 6]

2.3.2 Information-Theoretic Measures

A common problem of distance-based measures is that they imply *uniform* links, which means that a link always counts for the same distance, disregarding of the specificity the connected concepts. This is generally not appropriate, since links between more specific concepts should be considered *semantically* shorter than those between general concepts. The following section describes approaches of similarity measures that do not rely on edge counting, but on the information theoretic evaluation of an ontology [32, 34–37].

These measures derive the similarity of two concepts from the specificity (*information content*) of their commonalities. The specificity of a concept c is determined by the probability p to find an instance of that concept in the overall set of instances R. The higher this probability is, the lower is its *information content* and the more general it is considered to be. If the ontology has a unique root c_0, then $p(c_0) = 1$, meaning c_0 is the most unspecific class. The information content $I(c)$ of a class $c \in C$ is defined as

$$
\begin{aligned}
I(c) &= -\log(p(c)) \\
p(c) &= \frac{|R(c)|}{|R|} \\
R(c) &= \{i \in I \mid \exists\, c' \in C(c) : i \,(\text{typeOf})\, c\} \\
C(c) &= \left\{c' \in C \mid c \,(\text{subtypeOf})^* \, c'\right\}
\end{aligned}
$$

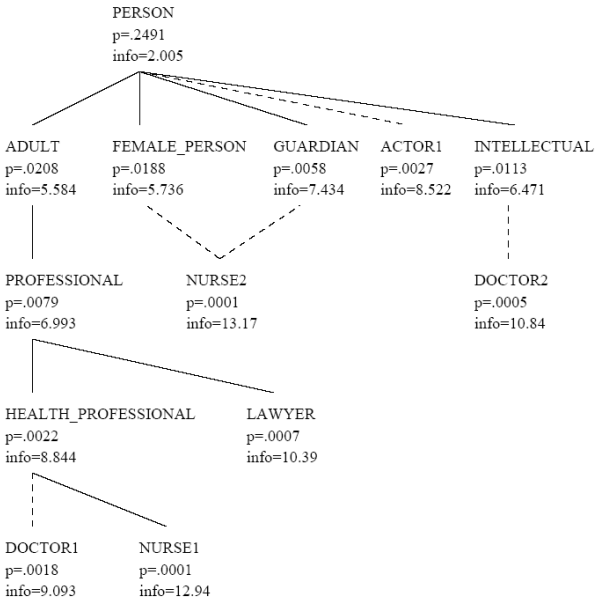


Figure 2.9: A fragment of the WordNet taxonomy with probabilities and information content assigned [32]

As an example, figure 2.9 shows a fragment of the WordNet taxonomy with assigned probability and information content values [32].

Lin

In his work "An information-theoretic definition of similarity" [34], Lin introduces a formal definition of concept similarity, which it is based on the following intuitions.

Intuition 1 The similarity between two concepts c_1 and c_2 is related to their commonalities. The more commonalities they share, the more similar they are.

Intuition 2 The similarity between two concepts c_1 and c_2 is related to the differences between them. The more differences they have, the less similar they are.

Intuition 3 The maximum similarity between two concepts c_1 and c_2 is reached when c_1 and c_2 are identical, no matter how much commonalities they share.

Beside these intuitions, Lin makes the following reasonable assumptions in order to define a general similarity measure:

Assumption 1 Let $I(common(A, B))$ be the measure of A's and B's commonalities, where $common(A, B)$ is a proposition that extracts A's and B's commonalities and $I(s)$ is the amount of information a proposition s has. If, for instance, A is "apple" and B is "orange", $common(A, B)$ would be $(fruit(A), fruit(B))$. In information theory, the information contained in a proposition is measured by the negative logarithm of its probability. Therefore, $I(common(A, B)) = -\log(P(fruit(A), fruit(B)))$.

Assumption 2 The differences δ between A and B are given by

$$\delta(A, B) = I(description(A, B)) - I(common(A, B))$$

where $description(A, B)$ is a proposition that states *all* information about A and B.

Assumption 3 The similarity $sim(A, B)$ between A and B is a function of their commonalities and differences: $sim(A, B) = f(I(common(A, B)), I(description(A, B)))$. The range of f is $\{(x, y)|x \geq 0, y \geq x\}$ and $f : \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \mapsto [0 \dots 1]$.

Assumption 4 As the similarity of a pair of identical concepts is 1, f must have the following property: $\forall x \geq 0 : f(x, x) = 1$.

Assumption 5 If the commonality between A and B is empty, their similarity must always be 0: $\forall x \geq 0 : f(0, y) = 0$.

Assumption 6 As objects generally can be viewed from different perspectives, their similarity can be computed separately for each perspective. It is assumed that the overall similarity is a weighted average of the similarities of all observed aspects. The weights are the amount of information of the aspect descriptions. This leads to

$$\forall x_1 \leq y_1, x_2 \leq y_2 : f(x_1 + x_2, y_1 + y_2) = \frac{y_1}{y_1 + y_2} f(x_1, y_1) + \frac{y_2}{y_1 + y_2} f(x_2, y_2)$$

Based on these assumptions, a general similarity measure definition can be derived:

$$\text{let } x = I(common(A, B)) \text{ and } y = I(description(A, B))$$

$$\begin{aligned} sim_{LIN}(A, B) &= f(x, y) \\ &= f(x + 0, x + (y - x)) \\ &= \frac{x}{y} \cdot f(x, x) + \frac{y - x}{y} \cdot f(0, y - x) \\ &= \frac{x}{y} \cdot 1 + \frac{y - x}{y} \cdot 0 = \frac{x}{y} \\ &= \frac{I(common(A, B))}{I(description(A, B))} \\ &= \frac{2 \cdot \log(P(lcs(A, B)))}{\log P(A) + \log P(B)} \end{aligned}$$

	sim_{Rada}	sim_{Cast}	sim_{Haas}	sim_{LC}	sim^{R}_{WP}	sim_{HO}	sim_{Lin}	sim_{Res}
increases with commonalities	-	-	+	-	+	-	+	+
decreases with differences	+	+	+	+	+	+	+	-
$sim(c,c) = max$	+	+	+	+	+	+	+	-
$sim(c_1,c_2) = sim(c_2,c_1)$	+	+	+	+	+	+	+	+
$max(sim) = 1$	+	+	+	+	+	-	+	+
supports any properties	-	+	-	-	-	+	*	*

* properties are not considered

Table 2.1: Feature comparison of semantic similarity measures

Resnick

Resnik also introduced an information-theoretic similarity measure, which simply considers the most informative subsumer (MIS) [32].

$$sim_{RES}(c_1,c_2) = \max_{c \in S(c_1,c_2)} (-\log p(c))$$

There, $S(c_1,c_2)$ is the set of concepts that subsume c_1 and c_2. The maximum similarity of this measure is the maximal information content of a concept $c \in C$, which is the concept with the lowest probability of finding an according individual $i \in I$. This maximum can be used to normalize the measure.

$$max_{sim_{RES}} = -\log \frac{1}{|I|} = \log |I|$$

2.3.3 Comparison

The intuitions of Lin provide a fair basis for a comparison of the previously discussed approaches. As graph distance reflects the differences of concepts, all distance-based measures tribute to Lin's second intuition. The ontological height of the LCS, on the other hand, reflects commonalities. Since all distance-based measures except sim_{Haas} do not take this height into account, they do not comply with Lin's intuition 1.

Regarding the information content based approaches, just the measure Lin meets all intuitions. Resnik considers the maximum information content of common subsumers. As they reflect commonalities, this measure does fulfill Lin's intuition 1. Nonetheless, possible differences are not reflected, since Resnik's formula does not consider the distance of the concepts to their *MIS*. Therefore, the measure of Resnik does not comply with intuition 2. A complete comparison of the key characteristics of the different approaches is given in table 2.1.

2.4 Swarm Intelligence

Swarm Intelligence denotes a field of artificial intelligence that investigates the behavior, the organization and the strategies of natural social societies, such as formed by insects, birds, fishes or humans, in order to adapt them for various kinds of computerized problem solving. These societies are characterized by a high frequency of local interactions between relatively simply agents, which do not follow a designed leader but act highly independent, local and asynchronously [38]. Though the interactions are very simple, social societies have proven to be capable of solving complex problems, like work balancing or routing, and, at the same time, to be highly scalable and adaptive to environmental changes. By cooperation of millions up to billions of individual beings, termites and ants, for example, form and maintain impressive structures (ant hills, termite mounds), coordinate food supply, maintain temperature and organize colony protection and attacks. The largest known single connected ant colony was found in South Europe spanning over 5760 kilometers from the Italian Riviera to the north west of Spain[4]. Being over 700 kilometers longer than the Chinese Wall, it is also the largest known building of any kind on earth.

One example of a highly distributed multi-agent system, which makes use of swarm intelligence, is the Ant Colony Optimization (ACO). It is used to obtain fast near-optimal solutions for shortest path problems of various kinds, like scheduling, sequential ordering, vehicle routing, personnel dispatch or the Traveling Salesperson Problem (TSP).

The ACO generally simulates ant behavior studied in the *double bridge experiment* [39]. There, ants were given two possible paths of different length from their nest to a source of food. They managed to always pick up the shorter path in the long run by constantly dropping pheromones on their way. While at first the two paths were picked randomly, the shorter path increased its pheromone concentration as ants returned earlier than those who chose the longer path. It was observed that the higher pheromone concentration animated following ants to pick the shorter path more frequently, thus increasing the concentration even more. Figure 2.10 illustrates a pseudo-code adaption of that behavior for a computerized solution of the TSP with virtual ants and pheromones.

The key characteristics of these biological collectives - *scalability*, *simplicity*, *robustness*, *decentralization* and *parallelism* - are also demands for distributed computing and are therefore being studied and adapted. The next chapter describes the approach of using swarm intelligence to implement a distributed RDF Triple Space with swarm-based semantic clustering.

[4] http://www.3sat.de/3sat.php?http://www.3sat.de/nano/news/31769/index.html, 18.09.2008

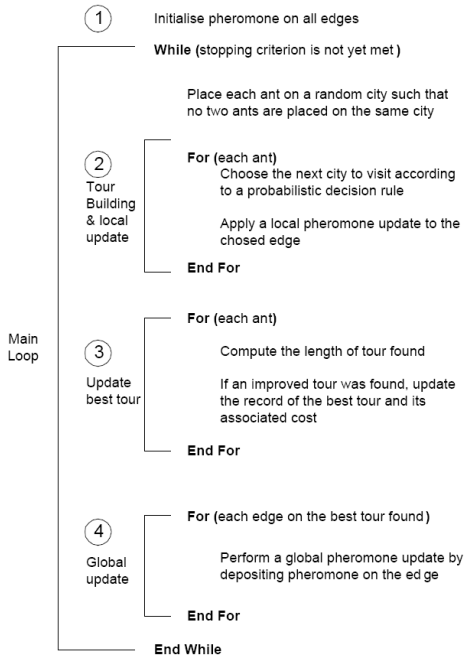


Figure 2.10: Pseudocode for a swarm-based solution of the TSP [39, page 6127]

2.5 SwarmLinda

SwarmLinda is a simulation of a distributed Tuple Spaces. It was implemented as a NetLogo-Extension by Daniel Graff in his master thesis at the *Freie Universität Berlin* in 2008 [2].

2.5.1 NetLogo

NetLogo is a general purpose network simulator developed by the Center for Connected Learning and Computer-Based Modeling of the Northwestern University, Chicago [40]. It provides mechanisms to build environments of graph- or patch-based worlds and allows to define application-specific patch-, graph- or link-properties. Moreover, it includes various visualization facilities such as a 2D- or 3D-View and flexible user defined plots, which can be used for activity and state monitoring. By using the NetLogo modeling language, it is possible to create different agents breeds as well as to implement their behavior in the environment. NetLogo updates all agents and the environment in a round-based way (using time ticks) and takes responsibility of the internal synchronization.

NetLogo is programmed in Java and allows to write custom extensions in Java as well, using the NetLogo Extension API. Bundled as a Java archive (JAR) file, these extensions can be imported and called as procedures from a regular NetLogo model. An overview of the most important applied features of NetLogo and their usage is given in section 4.2 (User Manual), page 70.

2.5.2 SwarmLinda Extensions

The NetLogo extensions and models implemented by Daniel Graff consist of a network generator and the actual Tuple Spaces simulation. The environment is defined by a graph, the nodes of which represent single tuple spaces and the edges of which represent the Tuple Space network connections. The basic IN and OUT primitives are performed by ant-agents that move tuples and templates between nodes, based on strictly local decisions. Along the way, they spread pheromones indicating which kind of tuple or template they carry. Next, the basic system functionalities are described. For further explanations on implementation details and evaluation results of the models and extensions, refer also to the original document [2]. Figure 2.11 shows a screenshot of the simulator main view.

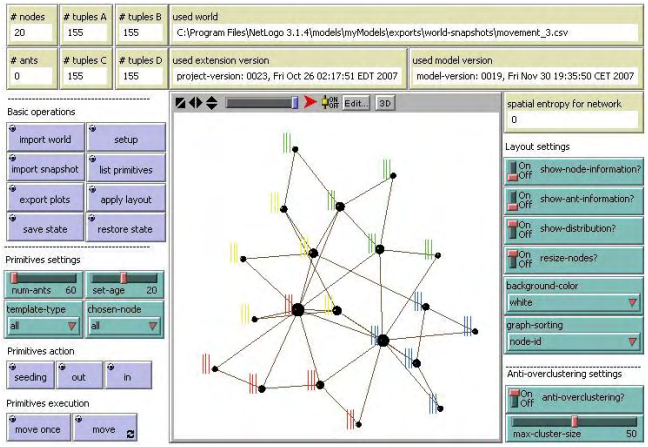


Figure 2.11: Control elements and visualization area of the SwarmLinda simulator [2, page 50]

OUT - Ants

When a tuple is stored in the space, an OUT-Ant is created which picks it up and tracks existing pheromones to find similar tuples to store it at that location. Following its path, the ant drops pheromones that match the types of its tuple on each visited node. Also, it gets older with each node transition it makes. If the ant decides to drop the tuple - because the current location seems suitable enough or because its *time-to-live* (*ttl*) became zero - it drops pheromones on the surrounding neighbors as well. After the ant has dropped its tuple, it dies in any case. The *drop probability* depends on the concentration c of matching tuples at the current location and the ant's time-to-live.

$$P_{drop} = \left(\frac{c}{c + ttl}\right)^2$$

The *path selection probability* from node i to node j is defined as

$$P_{i,j} = \frac{c_j + Ph_j}{\sum_{n \in NH(i)} (c_n + Ph_n)}$$

where $NH(i)$ states the neighborhood of node i and Ph_i the local amount of scent that matches the actual tuple or template.

IN / RD - Ants

Finding tuples for templates in the space works quite similar to tuple storage. At the requesting node, an IN- or RD-Ant gets born and follows the scents of its template to find a match. If it finds one at its current location, the matching tuple is returned to the requestor and, in case of an IN, removed. By returning on the visited path, the ant again spreads its pheromones. If there is no matching tuple present, the ant chooses one of the neighboring nodes the same way like OUT-Ants. IN- and RD-Ants also have a pre-defined *time-to-live* and age while traversing the network. If an ant dies, the request fails, assuming that the probability of finding a match at one of the unvisited nodes is very low.

Scout - Ants

Scout-Ants are created at random and move tuples to create a higher degree of order in the space. Technically, they are a combination of OUT- and IN-Ants, as they pick up a tuple which seems to be misplaced at its current location (this decision is based on the concentration of matching co-located tuples) and try to again store it in the space.

Conclusions

In his work, Daniel Graff shows how swarm intelligence can be used for collective sorting and clustering in Tuple Spaces. Although his system only supports limited kinds of tuples and templates, the achieved results successfully demonstrate how self organization, performed by agents making local decisions, can form a dynamic, adaptive, robust, decentralized and highly asynchronous LINDA implementation.

2.6 Related Work

2.6.1 JavaSpaces

JavaSpaces™ is a Sun Microsystems specification and reference implementation for the LINDA Tuple Spaces coordination model in the Java programming language. It is part of the Java Intelligent Network Infrastructure (JINI) Technology Core Platform Specification [41, 42]. In contrast to LINDA, JavaSpaces allows tuples (called *Entries*) to contain any serializable Java object and provides service implementations for accessing a single centralized Tuple Spaces (entry space). Those services are deployed in a conventional web server, where a minimal one is part of the project distribution [21].

JavaSpaces supports all the basic LINDA primitives (but EVAL) - although their names differ to match standard Java conventions (e.g. *read*, *write* and *take*). The matching of tuples with anti-tuples is done by byte-level comparisons [21], using a non-standard form of serialization of only public fields. JavaSpaces also extends the LINDA programming model with a concept of transactions, which is intended for commercial use, and an event registration mechanism to inform clients of tuple insertions. Furthermore, entries with the same values may exist multiple times in the space. Figure 2.12, which was taken from the Sun Microsystems JavaSpaces web page [41], illustrates a general application layout using JavaSpaces.

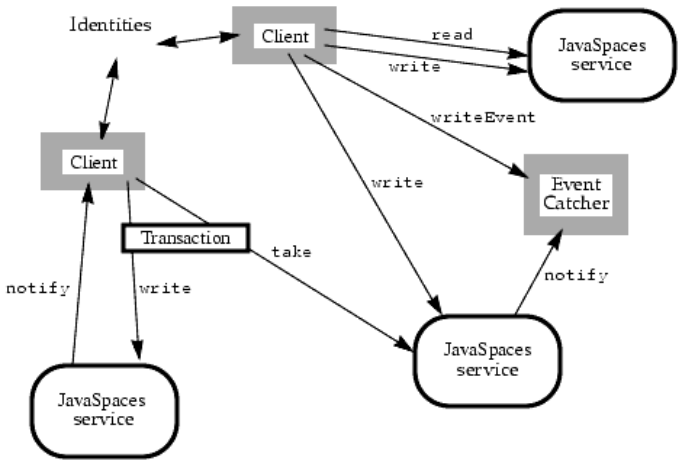


Figure 2.12: An illustration of a general JavaSpaces™ application. Clients perform operations that map entries or templates onto JavaSpaces services. These can be singleton operations (as with the upper client), or contained in transactions (as with the lower client) so that all or none of the operations take place. A single client can interact with as many spaces as it needs to. Identities are accessed from the security subsystem and passed as parameters to method invocations. Notifications go to event catchers, which may be clients themselves or proxies for a client (such as a store-and-forward mailbox) [41].

2.6.2 GigaSpaces

The GigaSpaces eXtreme Application Platform (GigaSpaces XAP) is a commercial implementation and extension of the JavaSpaces specification. Among the extensions, there are operations on multiple tuples like updating, deleting, or tuple counting. It is designed as a scalable application server and does not only provide a Java API, but also a framework for .NET and C++. It supports a SOAP-over-HTTP access for other non-Java clients as well.

The GigaSpaces XAP core Java development framework is called OpenSpaces and is published as open source. OpenSpaces is supposed to simplify the development and deployment of distributed applications on top of the GigaSpaces XAP run-time platform: "OpenSpaces supports several of the popular Java development frameworks, such as Spring, Mule and Hibernate. This integration enables seamless high availability and scalability for applications written in using these frameworks. Using OpenSpaces for example, Spring applications can seamlessly plug-in GigaSpaces XAP's remoting, messaging and data components to achieve high availability and scalability" [43].

Besides the extensions mentioned above, GigaSpaces functionality is quite similar to JavaSpaces as they implement the same specification [21, 43].

2.6.3 TSpaces

TSpaces is a product of IBM and based on Java technology as well. It is focussed on integrating database functionality to tuple spaces [44]. Like in JavaSpaces, TSpaces extend and rename the original Tuple Space primitives and establish methods like *delete*, *scan*, *countN*, *consumingScan*, *deleteAll*, *multiWrite* or *multiUpdate*. Furthermore, tuples can be addressed by a primary key (tuple id) - a common database concept - via methods like *readTupleById* or *deleteTupleById* [21]. Also there is an event notification mechanism for insertions and deletions of tuples and these tuples may contain any Java object. Unlike in JavaSpaces, tuple-template-matching is performed using the *equals()* or *compare()* methods of the contained objects (the latter in case *java.lang.Comparable* is implemented) instead of byte-level comparisons.

As in most commercial space implementations, TSpaces does support transactional space access and enforces security by a concept of users, user groups, rights and password protection. Another interesting feature of TSpaces is its aging mechanism, where tuples can have an expiring time after which they automatically vanish from the space.

2.6.4 XMLSpaces.NET

XMLSpaces.NET is a LINDA implementation in the Microsoft .NET framework that adds support for XML [45]. In XMLSpaces.NET, tuples and fields are modeled in XML, which allows a complex tuple structure, possibly containing sub-tuples. It implements

sophisticated matching enhancements, which evaluate the XML content, and allows the usage of XML query languages like XQuery [46] or XPath [47]. XMLSpaces.NET also includes support for a fully distributed space.

2.6.5 eLinda

eLinda is another Java implementation of the original LINDA programming model. The three major extensions of eLinda are a broadcast output operation, support for multimedia applications, and a Programmable Matching Engine that allows to use flexible criteria for tuple-template-matching [21]. This engine can, for instance, be used to find a tuple with a value closest to a given one or within a given range, or to evaluate tuple aggregations. It is also possible to define custom matchers with the help of the eLinda framework (for instance to evaluate XML content like in XMLSpaces.NET).

2.6.6 Tripcom Project

The Tripcom Project started in April 2006 and is currently planned to run until March 2009. It is a cooperation of 9 partners, situated in 7 different countries (Freie Universität Berlin, National University of Ireland, Technische Universität Wien, and Universität Stuttgart, just to mention some). Its objectives are to investigate how the combination of technologies like web ontologies and web services can form semantic web services, or how web ontologies and Tuple Spaces can form semantic Tuple Spaces. Altogether, these technologies can evolve into the next generation of the Semantic Web, which includes secure publication of semantic data, resource retrieval by semantic matching and trustful semantic mediation between heterogeneous services [22, 48, 49].

Chapter 3

Approach

In this chapter, the major research objectives of this thesis are presented. It contains analyses on the previous works of Daniel Graff [2] and Anne Augustin [7] and provides an outline of the proposed SwarmLinda extensions to form a semantically clustered Triple Space. Furthermore, all relevant aspects of the introduced swarm approaches are discussed in detail through the next sections.

3.1 Previous Works and Objectives

In his Master Thesis [2], Daniel Graff developed a swarm-based LINDA implementation, where tuple clustering and retrieval is accomplished by simulated ants using virtual pheromones for path selection and making strictly local decisions, like whether to drop a tuple at the current node or to transit to the next one (see section 2.5 (SwarmLinda)). There, the tuple drop probability as well as the node selection probability depend on the concentration of matches in the local set of tuples or in the scent-lists respectively. In his extension, Graff supports the usage of four explicit tuple kinds[1] only. By using a separate scent for each possible template, the maximal length of the scent-lists is limited to four as well. Consequentially, also concentration and entropy calculations presume the usage of just these distinct tuple kinds. Swarm-based approaches for a general semantic triple clustering and retrieval can evidently not rely on such restrictions.

Instead of just simple basic types, in the field of typed RDF Tuple Spaces, there exist semantically interlinked classes and properties that form ontologies. Instances of classes (individuals) can be instances of other classes at the same time, and moreover, these classes may subclass an arbitrary number of other classes themselves. Therefore, a typed matching of resources must consider the underlying ontological structure and derive the semantic similarity and relatedness of the concepts. As a result, these matchings can, unlike in SwarmLinda, not be modeled in a discrete or disjunct

[1] (*string*,*string*), (*string*,*int*), (*int*,*string*), and (*int*,*int*)

way. Instead, they are generally overlapping and represented as fuzzy values. This thesis introduces extended and generalized approaches for concentration, probability, and entropy calculations, which take multi-instancing, multi-inheritance, class- and property-similarity, and overlapping matches into account.

Daniel Graff's work is extended in the bachelor thesis [7] of Anne Augustin, who implements resource clustering by URI. Therefore, she analyzed namespace hierarchies of the most common URI schemes and introduced an adequate similarity measure for namespace comparison. In her extension, she establishes a *tripled* clustering, indexing triples by their subjects, predicates and objects.

Since this approach groups resources by similar URIs, it does not evaluate their semantic context and accordingly disregards the information provided by the ontologies. As URIs generally do not reflect the classes their resources belong to, triples clustered in this manner cannot efficiently be retrieved by type. In the next sections, swarm-based approaches are introduced, which provide support for typed templates (allowing typed triple retrieval), as they cluster statements semantically and form thematically confined areas within the Triple Space.

3.2 Overview

This section provides a brief outline of the approached strategies and depicts the Triple Space architecture. Regarding to the different abstraction levels of RDF ontologies (see section 2.1.1 (RDF) and 2.3.1 (Distance-Based Measures)), the clustering is divided into the *A-Box-* and the *T-Box Clustering*.

3.2.1 A-Box Clustering

The A-Box of an ontology contains all statements about class instances and values. *A-Box clustering* addresses the adaptive and type-based distribution of these statements among the Triple Space network. To gain web-scalability, every node of the Triple Space must only be responsible for A-Box statements referring to a subset of all existing resources. *Semantic clustering* implies that the responsibility for similar types must be assigned to nodes within the same network area. Furthermore, the assignment of this responsibility must be dynamic and adaptive to ontological and environmental changes (mainly changes in network topology).

These goals are achieved by utilizing the self-energizing aspects of swarm intelligence. The conceptual responsibility of a node can be dynamically derived from the type distribution of its triple resources. Using an appropriate similarity measure, the *semantic concentration* of a given type in this distribution can be calculated. Since this concentration can indicate the semantic suitablity of a node towards the resource of a passing ant, it is used to decide, whether to drop the triple at the current node or to continue the search.

Thus, a high concentration of a certain type leads to a increased probability of aggregating statements about similar resources, whereby the concentration increases even more.

Similar to the ACO, the locating of suitable nodes is supported by a system of semantic trails. These trails are formed dynamically by ants spreading and following typed pheromones. Analogously to the drop probability, the path selection probability is derived from the semantic concentration of a given type within these pheromones.

The A-Box-Cluster consists of three different incarnations, in which all triples are indexed by their subject, their predicate, and their object respectively. Hence, anytime a triple is inserted into the cluster, an OUT-Ant is created for each statement resource (subject, predicate and object) and being placed in the corresponding cluster incarnation. Each OUT-Ant carries the original statement, an indicator of which resource it is responsible for, and a local type hierarchy. This hierarchy initially consists only of the types of its primary resource. While an ant traverses the network, it learns more about its resources, by merging its own type hierarchy with the hierarchies of the visited nodes. Moreover, ants on the same node may also learn from each other by merging their type hierarchies as well. This refining knowledge is then used to determine the semantic suitablity of the current node. If the local triples are similar to the carried statement, the ants adds it to the local A-Box and returns. Otherwise, it decides which neighbor to pick next, depending on the present scents.

3.2.2 T-Box Clustering

The T-Box consists of two different sub-levels - the *RDF-S-Level* and the *Schema-Level*. As the *RDF-S-Level* is a pre-defined fixed set of triples, it is not necessary to deal with RDF-S-Level insertions or deletions. Hence, its triples are available by default on each node for general ontology processing.

The *Schema-Level* contains all custom class and property definitions. It furthermore defines their relations, describes their meanings and is therefore the basic source of information about resource similarity and relatedness. Distributing the *Schema-Level* among the Triple Space nodes is essential, as a web-scalable system cannot provide the details of all ontologies on every node. Yet, *Schema-Level clustering* must provide each node with a *partial* knowledge that is rich enough to understand the semantics of its resources.

A complete class or property definition in RDF does not only consist of a single triple defining the class or property itself, but also of those defining subclass/subproperty-relations as well as those assigning possible members. Hence, the member definitions with restrictions, ranges and domains are also needed as are the definitions of ranges and domains as well and so on. As a result, a full class or property definition can hardly be limited to a few triples, but is instead highly transitively interlinked with other class

and property definitions. A complete partitioning of the T-Box among the network is therefore rather impracticable.

Therefore, the distributed T-Box is restricted to the very definitions that are necessary for similarity determination, while all other Schema-Level statements are clustered like regular A-Box statements. In the following, this similarity-relevant T-Box subset will be referred to as *Schema-Box* (S-Box). The statements required for similarity determination are those defining resources as classes or properties and those expressing subclass and subproperty relations.

To provide the necessary partial ontology knowledge of a node, which is defined by the types of the hosted resources and the types in the scent-lists, some S-Box definitions must be available on more than one node. For this purpose, these definitions are partially replicated on-the-fly by the active ants, which automatically extend the local S-Box of a node with the (ant-local) type hierarchy of their resources, anytime they drop a triple or spread their pheromones. To prevent the node-local S-Boxes from growing too large, they are periodically cleaned. This cleaning removes all definitions that are not needed any longer - because statements have been removed or relocated, or corresponding scents have vanished.

3.2.3 Triple Retrieval

The triple retrieval is performed by IN-Ants, which use the resource types of their templates and the previously spread scents to find matching statements. Besides templates containing plain resources, which are also supported in Augustin's implementation, this work additionally allows *typed templates*. A special URI-scheme *rdf-typed://* is used to mark a triple resource as *typed request*, where the scheme-specific part consists the original resource URI.

Since there are three cluster incarnations, template matchings can be conducted concurrently by subject-, predicate- and object- IN-Ants, which independently follow the types of their resource. In case of competing deletes, a special distributed locking is performed by *Locking Ants*, which is described in section 3.8 (Triple Retrieval), page 61.

Regarding templates that contain *only* plain resources, it would be possible to re-use the approach of Anne Augustin. Yet, this work uses URI-clustering for the *resource type identification* step only (which is described in the next section). The first reason is that, in this manner, the cluster can handle all requests consistently and execute plain request semantically in the same way as typed and mixed requests. More importantly, anticipating future works, a semantic processing of plain resources allows possible fuzzy searches, which can retrieve triples within a certain similarity range of a given resource.

3.2.4 Triple Space Architecture

The Triple Space consists of two different cluster layers - a reduced version of the URI cluster of Anne Augustin, which is restricted to storing resource type, subclass, and subproperty definitions, and the Semantic Cluster, where all triples are stored in semantic neighborhoods, as shown in figure 3.1.

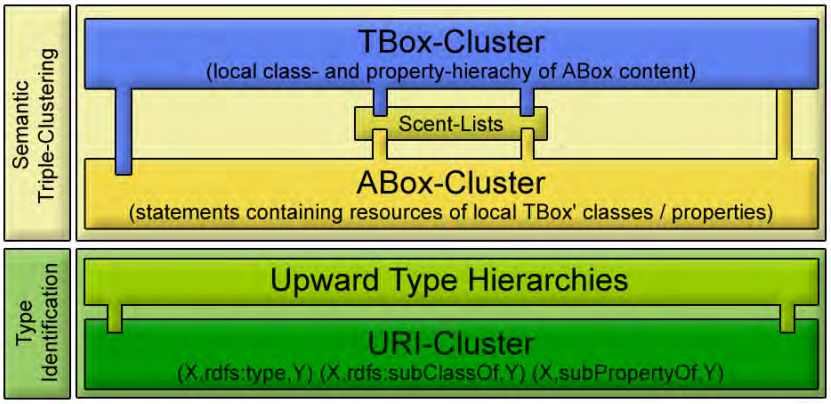


Figure 3.1: The triple Space architecture

The Semantic Cluster is a network of nodes, where each node for each cluster incarnation contains an A-Box, an S-Box, and a scent-list for each neighboring node. To operate the Triple Space, every node runs the following set of processes:

Triple and Template Processing This process continuously takes new triples and templates from the pending triple list of the node and creates the appropriate IN-, OUT- and RD-Ants. Having a pending list and a configurable maximum ant count prevents the Triple Space from ant cascades and provides a facility for activity balancing.

Cluster Maintenance The cluster is continuously maintained by this process, which picks up the local most dissimilar statements and creates OUT-Ants to relocate them. This increases clustering quality by decreasing the spatial sematic entropy of the Triple Space.

Template Generation In order to maintain scents and keep the cluster adaptive, this process creates random templates for present statements and assigns them to special RD-Ants, which first perform a random walk and then try to retrieve a match from their random starting point. By spreading pheromones like regular IN- or RD-Ants, they ensure that the scents are kept up-to-date, even if there are no active external requests (in which case all scents would completely vanish after a certain amount of time).

Pheromone Decrease This process continuously decreases the amount of pheromones in the local scent-lists by a configurable degree.

Garbage Collection As the states of the triple distributions and scent-lists may change permanently, this process removes any definitions in the local S-Box, which are not related to local resources or scent-list entries any longer. This ensures that the S-Boxes are kept as small as possible and contain only the relevant subset of classes and properties.

This concludes the general system outline. The following sections provide in-depth explanations of each aspect describing all used assumptions, strategies and calculations.

3.3 Type Identification

In order to cluster and retrieve triples by the types of their resources, these types must be identified prior to any subsequent activities. As initially a resource, represented as URI, does not contain any information about its semantic context, the type identification is conducted using URI-clustering. For this purpose, the URI-Cluster is limited to storing type relevant triples only, which are those having a predicate of *rdf:type*, *rdfs:subClassOf*, or *rdfs:subPropertyOf*. As the relevant resource in these triples is always the subject, the predicate and object index of the URI-Cluster are omitted and only its subject incarnation is used. Anytime a type-relevant triple is added or deleted, the URI-Cluster is updated as well.

The type identification of any triple or template is performed using regular URI-RD-Ants. For each resource *r*, these ants identify its types by following the URI of *r* in the subject cluster, looking for triples matching the template (*r*,?,?). The returned matches allow to determine whether the resource is a class (having (*r*,*rdf:type*,*rdfs:Class*) in the result set), a property (having (*r*,*rdf:type*,*rdf:Property*) in the result set) or an individual (otherwise). In case the resource is a class or property, its direct super classes or super properties are identified by processing the result set for matches of (*r*,*rdfs:subClassOf*,?) or (*R*,*rdfs:subPropertyOf*,?). If the resource is an instance of a class, its direct parent classes are determined by examining the matches for (*R*,*rdf:type*,?). Figure 3.2 illustrates a triple type identification achieved concurrently by three different subject RD-Ants.

Using the URI-Cluster for type identification in the described way does not necessarily provide a full transitive type hierarchy. This would require to create RD-Ants repeatedly, which is considered to be too inefficient. Instead, building up the complete upwards type hierarchy, which is needed for similarity determination, is accomplished in two ways. First, URI-Ants are created sporadically to search for referred super class definitions. These definition are then replicated at the node, which hosts the subclass. Additionally, the ants' learning capability allows them to complete their knowledge about their resources dynamically while traversing the cluster.

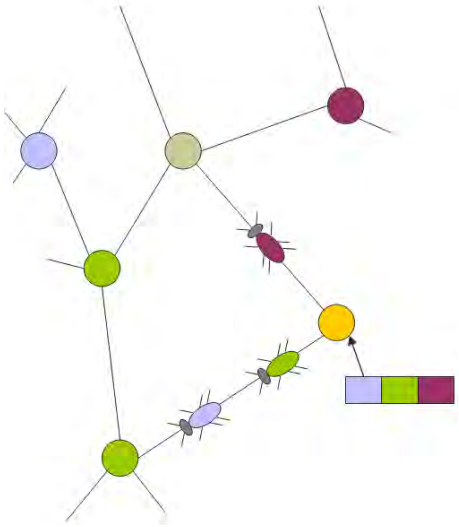


Figure 3.2: Using URI-clustering for type identification

3.4 Similarity Measure Evaluation

The most crucial factor of semantic clustering is the determination of type similarity. In order to select appropriate similarity measures for the cluster implementation, their different characteristics are compared and pre-evaluated in this section. As their gradients depend on various factors like concept distance, the height of the least common subsumer (LCS), or the information content of the most informative subsumer (MIS), the analysis is performed on three different types of ontology graphs, which are shown in figure 3.3.

There, example 1 is a graph with a low height and a high average degree containing 92 classes, example 2 is a graph with a high depth and a low average degree containing 38 classes and example 3 is a mixture of both containing 64 classes. These graphs are considered be representative enough for typical ontologies to perform an appropriate evaluation of the given measures. In these evaluations, Haase-Siebers-Harmelen was configured with $\alpha = 0.2, \beta = 0.6$ and, to meet the requirements of the information-theoretic measures, a number of 10 instances was assigned to each class.

Figure 3.4 shows the average similarity results of the evaluated measures[2] for each graph, grouped by concept distance. It can be observed that, although all measures decrease with graph distance, their actual results differ quite much.

[2]The following measures were omitted in this evaluation: the measure of Hirst and St-Onge because of its counterintuitive path direction restrictions, the measure of Castano, Ferrara, Montanelli and Racca because it is not intended to add an affinity weight facility to the Triple Space, and the measure of Jiang and Conrath because it cannot be normalized

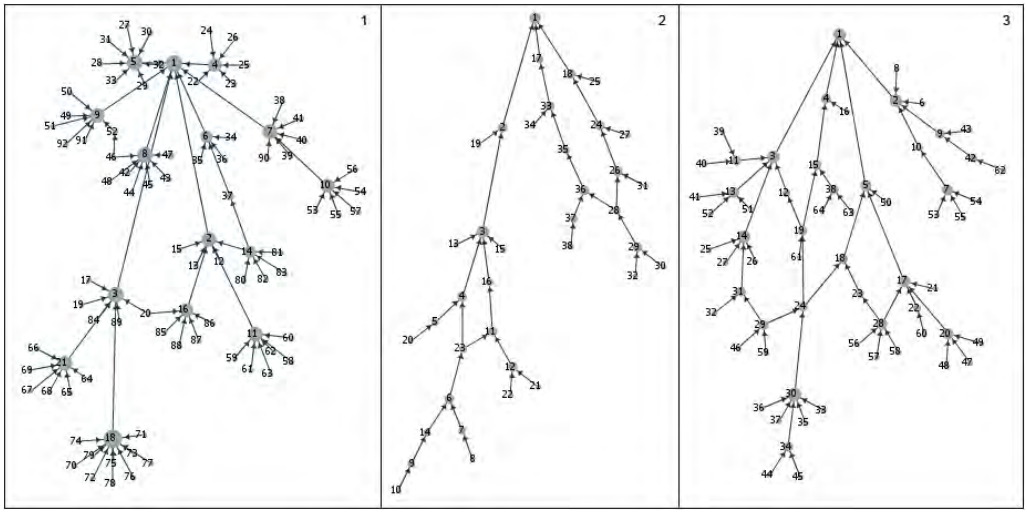


Figure 3.3: The example graphs used for similarity measure evaluation

3.4.1 Distance-Based Measures

At small distances (0-5), Rada-Mili-Bicknell-Blettner gives the lowest similarity results, as it initially decreases very fast. Wu-Palmer states the highest values, while Haase-Siebers-Harmelen's similarity is generally closest to the average. Rada-Mili-Bicknell-Blettner is also the most stable measure as it does not depend on the LCS height and gives the same results in all example graphs. While Wu-Palmer and Haase-Siebers-Harmelen decrease slower than Rada-Mili-Bicknell-Blettner at small distances, the situation changes at higher distances (>5). Furthermore, as their decreases do depend on the LCS height, they also differ among the example graphs and become more flat the deeper the graph is (see figure 3.4).

A second measure being unaware of the LCS height is Leacock-Chodorow, yet this measure is normalized by the diameter of the ontology graph and thereby decreases slower with an increased overall height. Moreover, it decreases generally slower than Rada-Mili-Bicknell-Blettner because of its logarithmic nature.

The overall average similarity for each graph and measure is displayed in figure 3.6. Surprisingly, while all other measures show a decrease of average similarity with increasing ontology height, Rada-Mili-Bicknell-Blettner does not. It even posts the highest average similarity at example graph 1, which actually has the lowest height. This is due to the fact, that Rada-Mili-Bicknell-Blettner is yet distance-, but not height-based and example 1 has the highest average degree, which results in the lowest average concept distance.

Wu-Palmer and Haase-Siebers-Harmelen tend to be quite similar, as their similarities progress very much alike within the observed cases (where the results of Wu-Palmerare slightly higher than those of Haase-Siebers-Harmelen). Figure 3.5 shows a comparison of the different similarity decreases of among the example graphs for each measure.

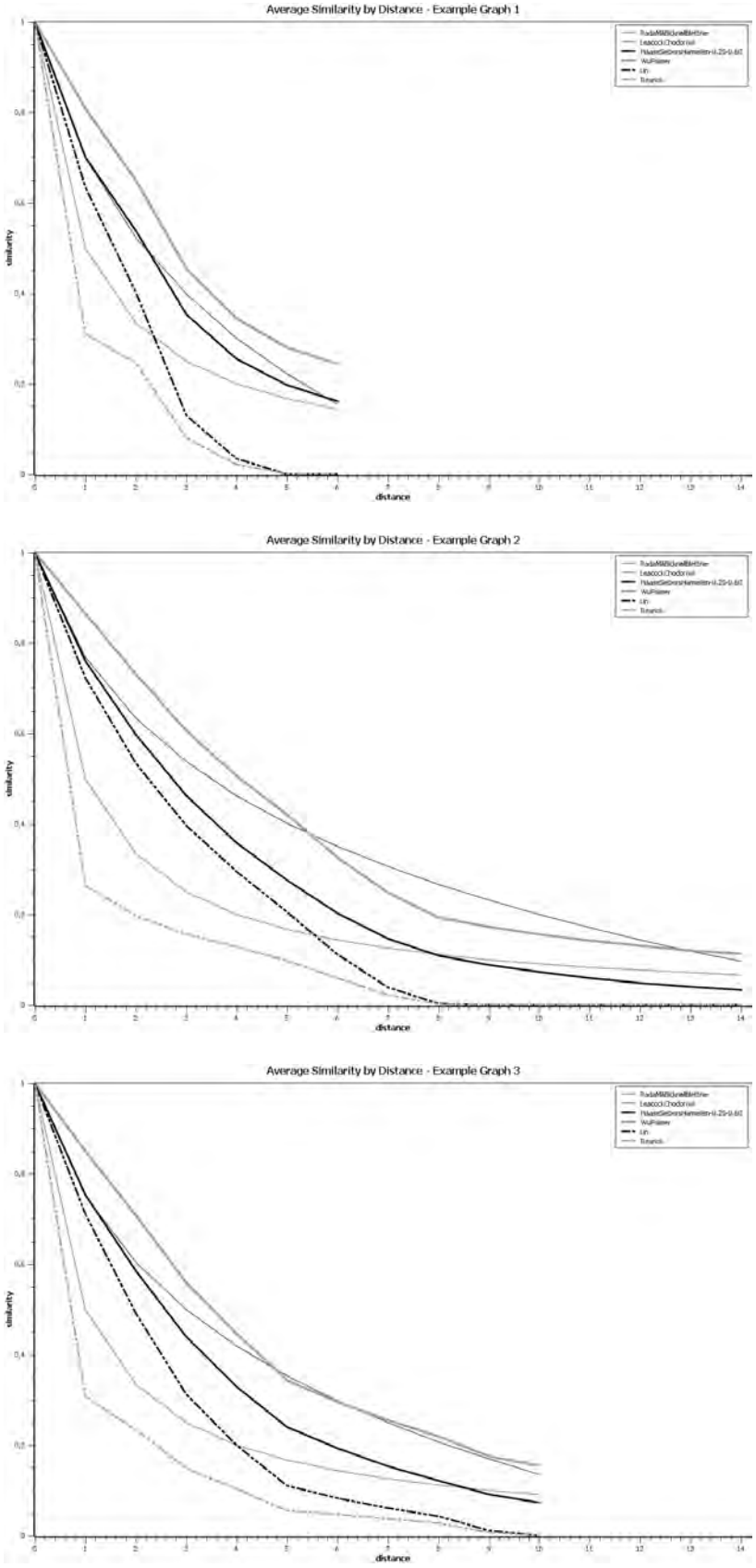


Figure 3.4: Average similarity results for example graphs

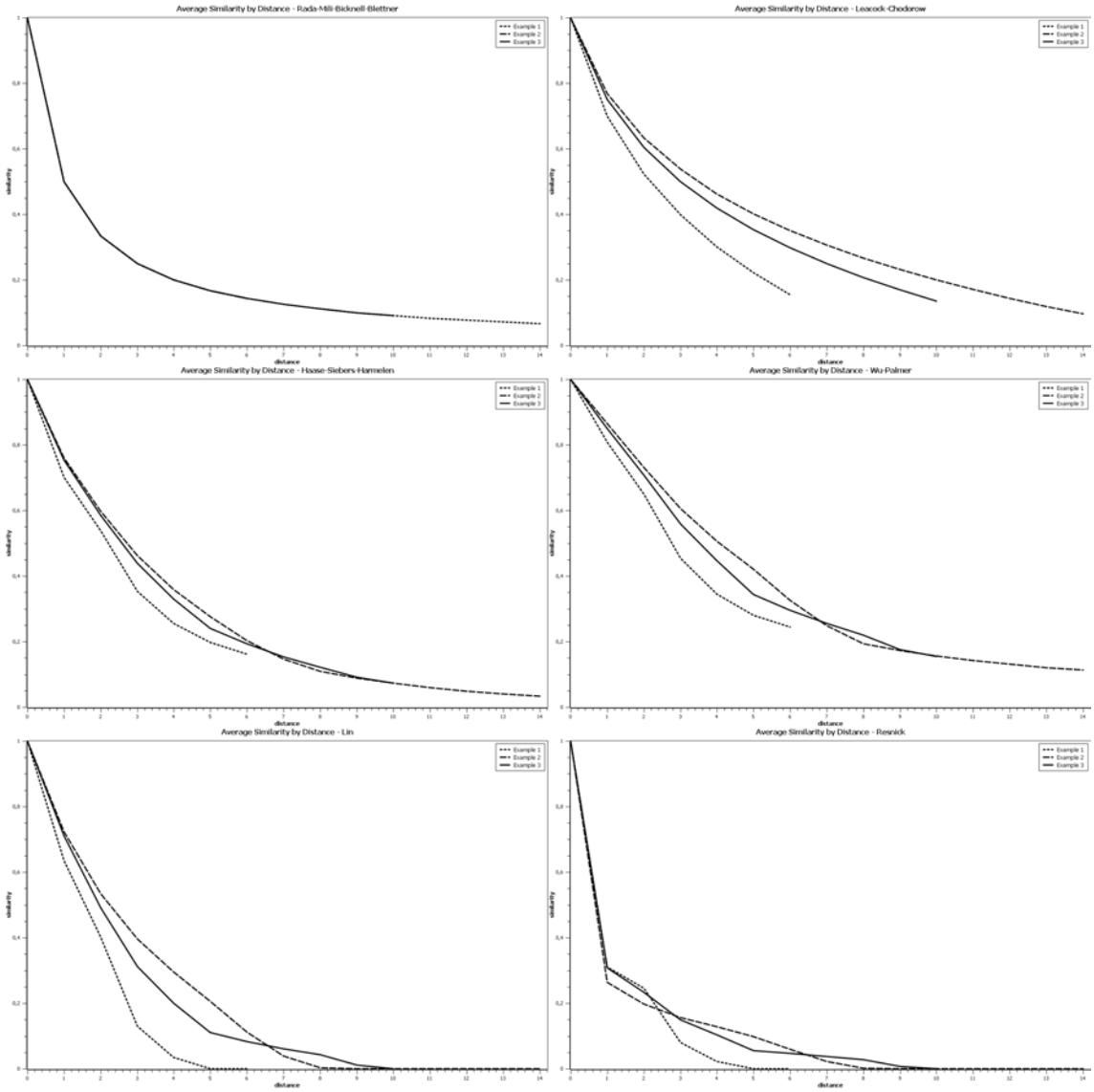


Figure 3.5: Similarity measure decrease in example graphs

3.4.2 Information-Theoretic Measures

A pre-evaluation of the information-theoretic measures of Lin and Resnick is not that straightforward, because the information content is quite independent of concept height or distance. Instead, the determining factor is the distribution of instances, which cannot generally be predicted. Therefore, the analysis in the example graphs, where individuals are equally distributed, can only be considered as a hint for the general behavior of these measures, but not to be representative.

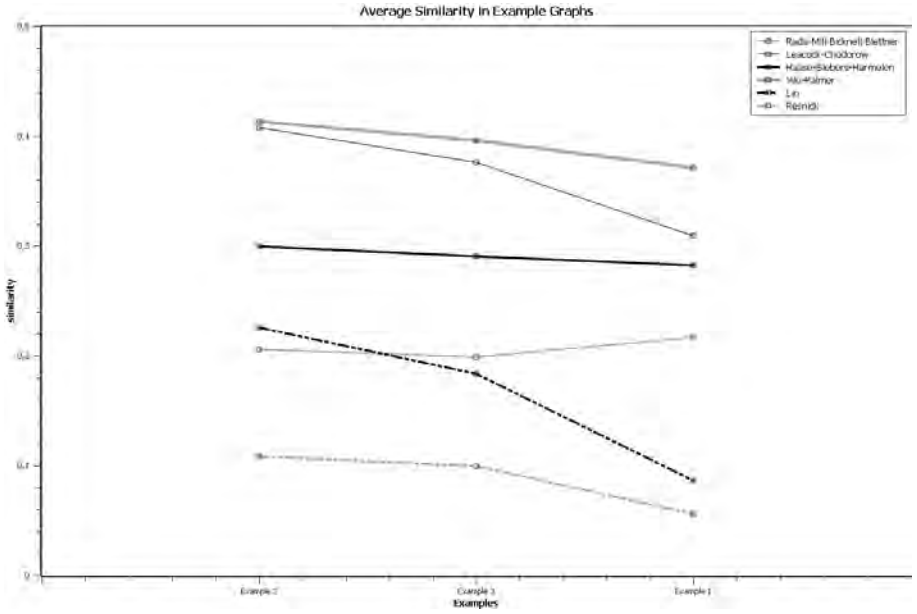


Figure 3.6: Average similarity in example graphs

3.4.3 Conclusion

Considering the made observations, the following measures are used or rejected for the Semantic Cluster implementation:

- **Rada-Mili-Bicknell-Blettner** is not used because it does not fully comply with similarity intuitions, as it does not increase with commonalities and may designate an increased average similarity in graphs with high degrees. In a clustering scenario this is a major disadvantage as a high average similarity between concepts may thwart clustering quality.
- **Haase-Siebers-Harmelen** is used because it fully complies with similarity intuitions and its results are generally closest to the average distance-based similarity. Besides, it provides the scaling variables α and β which allow to control the influence of height and distance independently.

- **Wu-Palmer** is not used because its gradient is rather similar to Haase-Siebers-Harmelen. In fact, a fair approximation of Wu-Palmer can be achieved using Haase-Siebers-Harmelen with $\alpha = 0.14$ and $\beta = 0.7$, which is shown in figure 3.7.
- **Leacock-Chodorow** is used as a second distance-based measure as it almost fully complies with similarity intuitions and states an upper bound of similarity in comparison with Haase-Siebers-Harmelen and the information-theoretic measures.
- **Lin** as information-theoretic measure Lin is used because it also fully complies with similarity intuitions and its results are in between those of Resnick and the distance-based approaches.
- **Resnick** is not used as it does not fully comply with similarity intuitions and generally states very low similarities compared to the other measures.

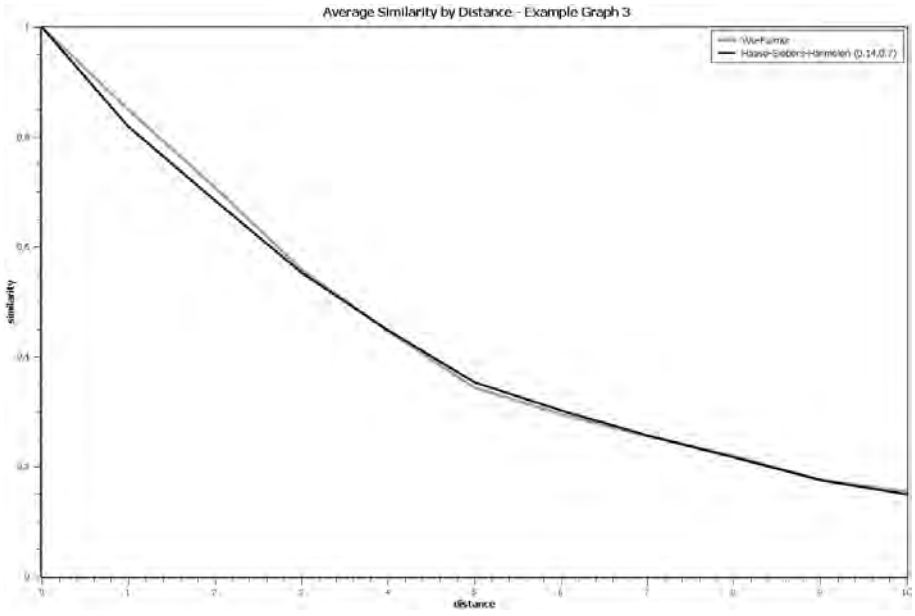


Figure 3.7: Wu-Palmer-approximation using Haase-Siebers-Harmelen

3.4.4 Measure Modifications

Degree Normalization

Figure 3.8 shows the accumulated similarity results (normalized, so that the values sum up to 1) for each measure and example graph, grouped by distance. Evidently, due to the fact that no similarity measure considers the *degree* of the nodes, these results do not have their maximum at zero distance. Since the amount of concepts within a certain distance generally increases faster than their similarity decreases, the accumulated similarity of concepts within distance 1, for instance, is in all cases higher than their self-similarity

(distance 0). As a result, when using these similarities in probabilistic decisions, like whether to drop a triple at the current node or to choose the next to visit, the overall probability of picking a node that hosts a concept within distance 1, might be higher than the probability of picking the node that hosts the concept itself. Therefore, an optional degree normalization can be applied in these calculations, which divides the original similarity by the number of concepts within the same distance:

$$sim_{normed}(c_1, c_2) = \frac{sim(c_1, c_2)}{|\{c_i | d(c_1, c_i) = d(c_1, c_2)\}|}$$

where $d(c_1, c_2)$ states the graph distance of the concepts.

Figure 3.9 shows the degree-normalized similarities of the chosen measures. After normalization, the overall similarity gradient has its maximum at distance zero and also decreases monotonically. The actual effect of degree normalization on clustering quality is evaluated in chapter 5 (Evaluation), page 85.

Measure Scaling

As the similarity distribution has a major influence on clustering, it is important to have a scaling facility. Using the unscaled measure might result in a rather too flat similarity distribution, stating too much average similarity between concepts for a proper clustering. Haase-Siebers-Harmelen already provides the scaling variables α and β, which control the influence of height and distance on similarity. A scaling Leacock-Chodorow and Lin is achieved by introducing the scaling variable γ and δ and applying them on the results as a scaling exponent: sim_{LC}^{γ} and sim_{LIN}^{δ}. Figure 3.9 shows how the appliance of $\alpha = 0.8$, $\beta = 0.6$, $\gamma = 32.0$, and $\delta = 3.0$ can be used to de-flatten similarity decrease.

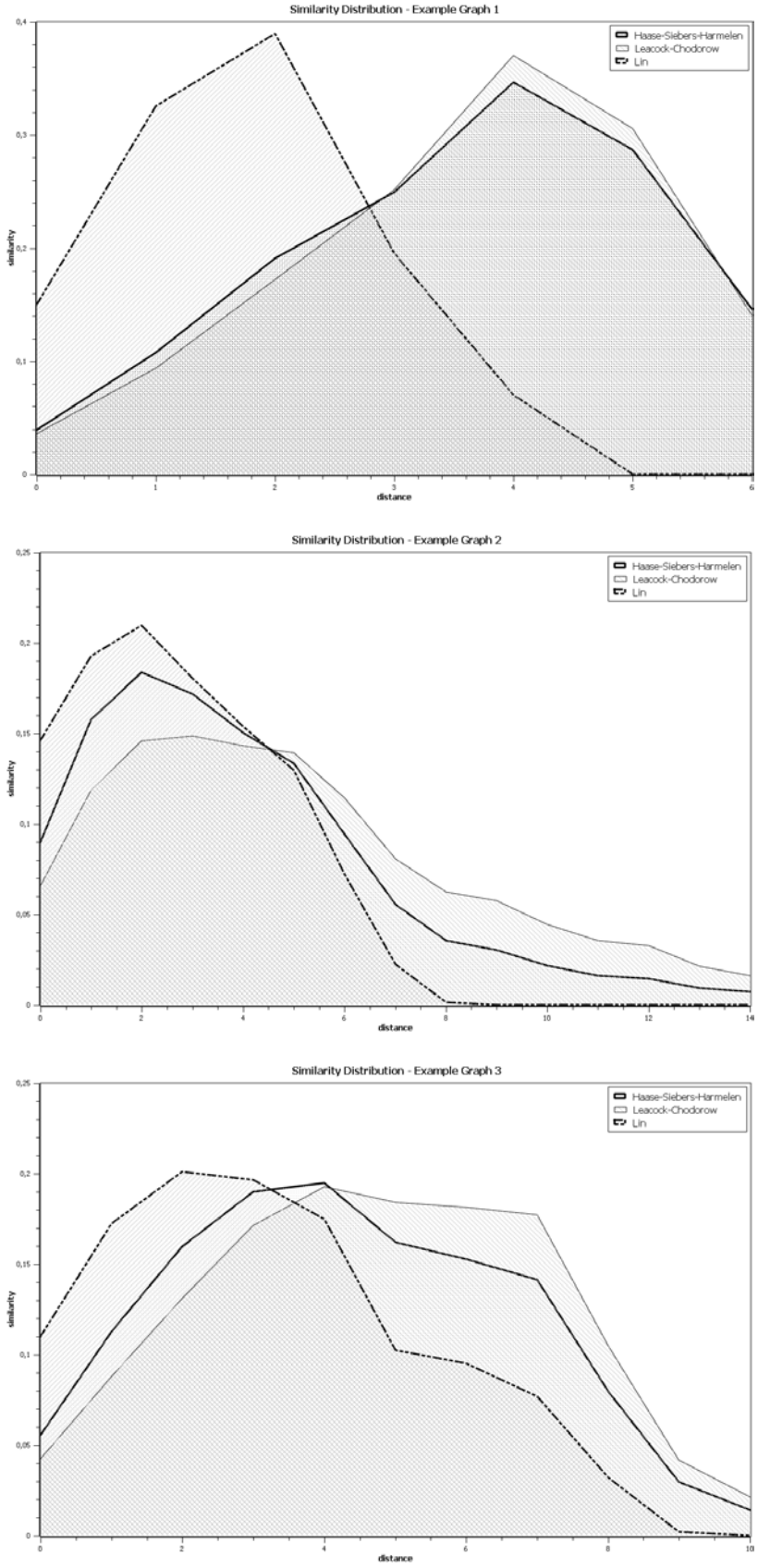


Figure 3.8: Similarity distribution in example graphs

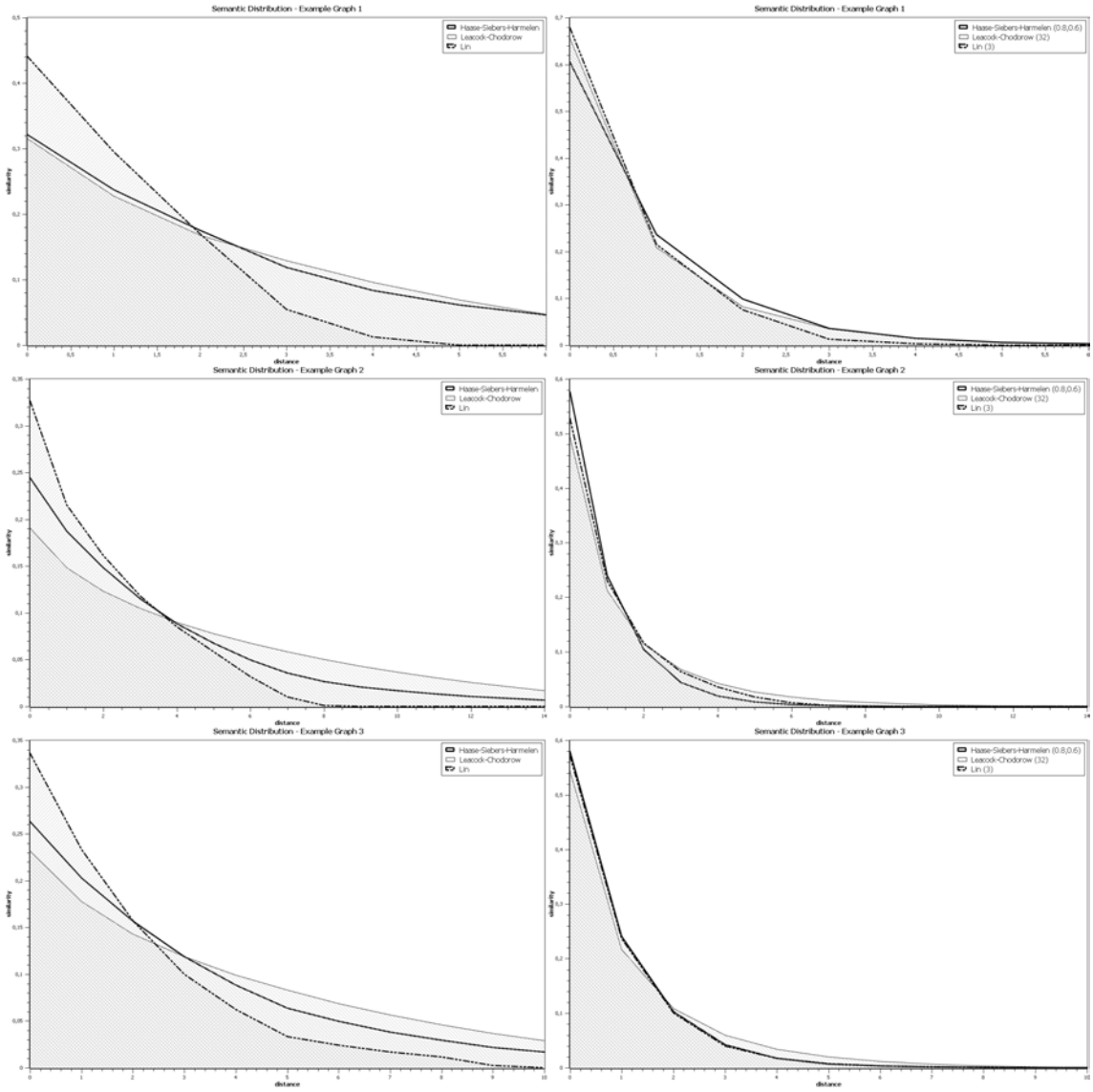


Figure 3.9: Similarity distribution in example graphs with applied degree-normalization

3.5 Notation

Next, the semantic clustering and retrieval are explained using the following notation:

symbol	meaning
$\mathcal{N}$	the set of Triple Space nodes
Θ	the overall un-clustered ontology
Ω	the set of triples (statements) in Θ
$\mathcal{K}$	the set of cluster indicators $\mathcal{K} = \{S, P, O\}$
$\mathcal{R}$	the set of resources in Θ
$\mathcal{R}^\kappa$	the set of resources appearing as subject, predicate or object (depending on $\kappa \in \mathcal{K}$)
$\mathcal{T}$	the set of type-resources (classes and properties) in Θ $r \in \mathcal{T} \Leftrightarrow (r, \textit{rdf:type}, \textit{rdfs:Class}) \in \Omega \vee (r, \textit{rdf:type}, \textit{rdf:Property}) \in \Omega \vee$ $\exists t \in \mathcal{T} : (r, \textit{rdfs:subClassOf}, t) \in \Omega \vee (r, \textit{rdfs:subPropertyOf}, t) \in \Omega$
$\mathcal{I}$	the set of all non-type-resources in Θ $\mathcal{I} = \mathcal{R} \setminus \mathcal{T}$
$\mathcal{A}$	the set of ants
$\mathcal{T}_t^\triangle$	the set of super-types of a type $t \in \mathcal{T}$ $t_i \in \mathcal{T}_t^\triangle \Leftrightarrow (t, \textit{rdfs:subClassOf}, t_i) \in \Omega \vee (t, \textit{rdf:Property}, t_i) \in \Omega \vee$ $\exists t_j \in \mathcal{T}_t^\triangle : (t_j, \textit{rdfs:subClassOf}, t_i) \in \Omega \vee (t_j, \textit{rdf:Property}, t_i) \in \Omega$
$\mathcal{T}_r$	the set of direct parent types of a resource $r \in \mathcal{R}$ $t \in \mathcal{T}_r \Leftrightarrow r \in \mathcal{I} \wedge (i, \textit{rdf:type}, t) \in \Omega \vee r \in \mathcal{T} \wedge r = t$
$\mathcal{T}_r^*$	the set of parent types of a resource $r \in \mathcal{R}$ $\mathcal{T}_r^* = \bigcup_{t \in \mathcal{T}_r} \{t\} \cup \mathcal{T}_t^\triangle$
${}^n\mathcal{N}$	the set of nodes neighboring a node $n \in \mathcal{N}$
$\mathcal{R}_t$	the set of resources in $\mathcal{R}$ having $t \in \mathcal{T}$ as direct parent type
$\mathcal{R}_t^*$	the set of resources in $\mathcal{R}$ having $t \in \mathcal{T}$ as parent type
$\mathcal{A}_O$,$\mathcal{A}_{IR}$	the set of OUT-Ants and the set of all IN- or RD-Ants
x_j	for any vector x, its value at index j is stated by x_j
x_j	for any matrix x, its column j is stated by x_j
x^i	for any matrix x, its row i is stated by x^i
x_j^i	for any matrix x, its value at row i and column j is stated by x_j^i

Furthermore, any restrictions of a variable or set X to a certain node $n \in \mathcal{N}$, ant $a \in \mathcal{A}$, cluster indicator $\kappa \in \mathcal{K}$, type $t \in \mathcal{T}$, resource $r \in \mathcal{R}$ or time τ are indicated as ${}^nX$,${}^aX$,X^κ, X_t,X_r, or ${}_\tau X$ respectively, and their combinations.

3.6 A-Box Clustering

3.6.1 Similarity Distribution

This section introduces the *similarity distribution* of a type as a measure of how its semantic similarity distributes over an ontology. It is derived from the regular or degree-normalized similarity of a type towards all other types $t_i \in \mathcal{T}$, which is calculated by any of the similarity measures described in section 2.3 (Semantic Similarity and Relatedness Measures), page 22. The similarity distribution is used by ants in probabilistic decisions - like whether to drop a triple or which node to pick next - and to determine the spatial sematic entropy of the Triple Space, which indicates the clustering quality.

Definition 3.1 (Similarity Distribution) *The* similarity distribution $simd_t$ *of a type* $t \in \mathcal{T}$ *is a vector containing the similarities of* t *to all other types. It is normalized by the total amount of similarity, so that its componentwise sum equals 1.*

$$simd_t = \left[\frac{sim(t, t_i)}{\sum_{t_j \in \mathcal{T}} sim(t, t_j)} \middle| t_i \in \mathcal{T} \right]$$

$$\forall t \in \mathcal{T} : \sum_{t_j \in \mathcal{T}} simd_t^{t_i} = 1 \tag{3.1}$$

Figure 3.8 shows the average similarity distribution of all types in the example ontologies used for similarity measure evaluation. There, the distribution is calculated without degree normalization, while figure 3.9 displays the results with degree normalization applied.

3.6.2 OUT-Ants

The semantic A-Box-Clustering is conducted by a swarm of OUT-Ants. In this swarm, every OUT-Ant retains this information:

- the assigned triple ω
- a cluster indicator $\kappa \in \mathcal{K}$, indicating which of the triple resources it is responsible for (this resource is also referred to as the *primary resource* or ω_κ)
- a local type hierarchy ${}^a\mathcal{T}$, representing the ant's ontological understanding
- the similarity distribution ${}^a simd$ for the types of its hierarchy
- its home-node ${}^a n$, its time-to-live ttl, and a path-memory

All ants' activities are divided into seven distinct phases, which are described next.

3.6.3 Initialization

This is the initial phase of newly created ants, where their statement ω and indicator κ are assigned as well as their type hierarchy ${}^{a}\mathcal{T}$, their home-node ${}^{a}n$, and their time-to-live *ttl*.

3.6.4 Learning

In this phase, the ant merges its own type hierarchy ${}^{a}\mathcal{T}$ with the type hierarchies of the current node $n \in \mathcal{N}$ and all other present[3] ants, gaining additional knowledge about the semantic context of its resources.

$$ {}^{a}_{\tau+1}\mathcal{T} = {}^{a}_{\tau}\mathcal{T} \cup {}^{n}_{\tau}\mathcal{T} \bigcup_{a_i \in {}^{n}\mathcal{A}} {}^{a_i}_{\tau}\mathcal{T} $$

When the merging is complete, the ant applies the configured similarity measure on its new knowledge and re-calculates its similarity distribution ${}^{a}simd$.

3.6.5 Task Fulfillment

In the task fulfillment phase, OUT-Ants determine the probability of dropping their statement at the current node. The drop probability arises from the concentration of the types of the ant's primary resource in the local *triple distribution* of the current node.

Definition 3.2 (Triple Distribution) *The* triple distribution ${}^{n}td^{\kappa}$ *of a node* $n \in \mathcal{N}$ *and a cluster indicator* $\kappa \in \mathcal{K}$ *is a vector which, for each type* $t \in {}^{n}\mathcal{T}$*, contains the number of triples* $\omega \in {}^{n}\Omega^{\kappa}$ *having a primary resource with* t *as direct parent type. If a primary resource has more than one direct parent types, the triple's count is equally divided among these types. Hence, the component-wise sum of the triple distribution equals the number of triples located at* n*.*

$$ {}^{n}td^{\kappa}_{t} = \sum_{\omega \in {}^{n}\Omega^{\kappa}} typecount(\omega, t, \kappa) $$

$$ typecount(\omega, t, \kappa) = \begin{cases} \dfrac{1}{\left|{}^{n}\mathcal{T}_{\omega_\kappa}\right|} & t \in {}^{n}\mathcal{T}_{\omega_\kappa} \\ 0 & otherwise \end{cases} $$

$$ \forall n \in \mathcal{N}, \kappa \in \mathcal{K} : \sum_{t \in {}^{n}\mathcal{T}} {}^{n}td^{\kappa}_{t} = \left|{}^{n}\Omega^{\kappa}\right| \tag{3.2} $$

Definition 3.3 (Type Concentration) *The* type concentration ${}^{n}C^{\kappa}_{t}$ *of a type* $t \in {}^{n}\mathcal{T}$ *at a node* $n \in \mathcal{N}$ *for a given cluster indicator* $\kappa \in \mathcal{K}$ *is the component-wise sum of the similarity-distribution-weighted values of the node's triple distribution* ${}^{n}td^{\kappa}$*, divided by the overall local triple count.*

[3] Technically, ants are considered to be present at a node until they arrive at another one.

$$ {}^{n}C_{t}^{\kappa} = \frac{\sum_{t_i \in {}^{n}\mathcal{T}} {}^{n}td_{t_i}^{\kappa} \cdot simd_{t}^{t_i}}{|{}^{n}\Omega^{\kappa}|} \quad , \quad |{}^{n}\Omega^{\kappa}| \neq 0 $$

Taking the local concentration of the types of its primary resource, the ant can determine the drop probability for its statement.

Definition 3.4 (Drop Probability) *The* drop probability ${}^{n}pd_{\omega}^{\kappa}$ *of a triple* $\omega \in \Omega$ *at a node* $n \in \mathcal{N}$ *for a cluster indicator* $\kappa \in \mathcal{K}$ *is the summed type concentration of all direct parent types of the primary triple resource, divided by the number of direct parent types. If there are no statements located at n or the ant's time-to-live ttl is zero, the drop probability is 1.*

$$ {}^{n}pd_{\omega}^{\kappa} = \begin{cases} \dfrac{\sum_{t \in {}^{a}\mathcal{T}_{\omega}^{\kappa}} {}^{n}C_{t}^{\kappa}}{\left|{}^{a}\mathcal{T}_{\omega}^{\kappa}\right|} & |{}^{n}\Omega^{\kappa}| \neq 0 \wedge ttl \neq 0 \\ 1 & otherwise \end{cases} $$

In order to randomize the ant's decision, the effective drop probbalility is calculated by adding a probabilistic component.

Definition 3.5 (Effective Drop Probbalility) *The* effective drop probbalility ${}^{n}pde_{\omega}^{\kappa}$ *is a randomization of the drop probability, which is configured by a system variable* $\varphi_d \in [0 \ldots 1]$.

$$ {}^{n}pde_{\omega}^{\kappa} = {}^{n}pd_{\omega}^{\kappa} \cdot (1 - \varphi_d) + 0.5 \cdot \varphi_d $$

In the extreme cases, the decision is made by considering the concentration only ($\varphi_d = 0$) or totally at random ($\varphi_d = 1$). If the ant decides to drop its statement at the current node, it furthermore integrates the type hierarchy for its resources into the type hierarchy of the current node and updates the scent-lists of the surrounding neighbors. The pheromone updates are similar to those described in section 3.6.7 (Transition).

$$ {}^{n}_{\tau+1}\Omega^{\kappa} = {}^{n}_{\tau}\Omega^{\kappa} \bigcup \{s\} \qquad {}^{n}_{\tau+1}\mathcal{T} = {}^{n}_{\tau}\mathcal{T} \cup \bigcup_{\kappa \in \mathcal{K}} {}^{a}_{\tau}\mathcal{T}^{*}_{\omega_\kappa} $$

3.6.6 Path Selection

Unless the ant drops its statement, it chooses the next node to visit depending on the entries in the scent-lists of the current node.

Definition 3.6 (Scent-List) *The* scent-list ${}^{n,n}sc^{\kappa}$ *at node* $n \in \mathcal{N}$ *for a node* $n_i \in {}^{n}\mathcal{N}$ *and a cluster indicator* $\kappa \in \mathcal{K}$ *is a vector which contains the pheromones spread by ants who previously made a node transition from n to* n_i*. The entry (scent) for type* $t \in \mathcal{T}$ *at node n for node* n_i *will be referred to as* ${}^{n_i,n}sc_{t}^{\kappa}$.

Based on the scent-list content, the ant evaluates the semantic suitablity of each neighbor.

Definition 3.7 (Semantic Suitablity) *The* semantic suitablity ${}^{n_i,n}stb_t^\kappa$ *for a type* $t \in \mathcal{T}$ *and a cluster indicator* $\kappa \in \mathcal{K}$ *from node* $n \in \mathcal{N}$ *to node* $n_i \in {}^n\mathcal{N}$ *is the sum of the similarity-distribution-weighted amounts of pheromones in the scent-list* ${}^{n_i,n}sc^\kappa$.

$$ {}^{n_i,n}stb_t^\kappa = \sum_{t_i \in {}^{n_i,n}sc^\kappa} {}^{n_i,n}sc_{t_i}^\kappa \cdot simd_t^{t_i} $$

The transition probabilities arise from the suitabilities of the neighbors, as defined next.

Definition 3.8 (Transition Probability) *The* transition probability ${}^{n_i,n}pt_\omega^\kappa$ *for a triple* $\omega \in \Omega$ *and a cluster indicator* $\kappa \in \mathcal{K}$ *from node* $n \in \mathcal{N}$ *to node* $n_i \in {}^n\mathcal{N}$ *is the summed suitablity of* n_i *towards all direct parent types of the primary triple resource* ω_κ*, relative to the overall suitablity of all neighbors.*

$$ {}^{n_i,n}pt_\omega^\kappa = \frac{\sum_{t \in {}^a\mathcal{T}_{\omega_\kappa}} {}^{n_i,n}stb_t^\kappa}{\sum_{n_j \in {}^n\mathcal{N}} \sum_{t \in {}^a\mathcal{T}_{\omega_\kappa}} {}^{n_j,n}stb_t^\kappa} $$

In case $\sum_{n_j \in {}^n\mathcal{N}} \sum_{t \in {}^a\mathcal{T}_{\omega_\kappa}} {}^{n_j,n}stb_t^\kappa = 0$*, the transition probability is* $\frac{1}{|{}^n\mathcal{N}|}$*, meaning that if there is no suitable neighbor, the ant picks the next node uniformly at random.*

Analogously to the drop probability, also the transition probability is extended by a configurable randomized component, which contributes to the adaptiveness of the Triple Space and allows ants to explore nodes that have not been previously visited by similar ants.

Definition 3.9 (Effective Transition Probability) *The* effective transition probability ${}^{n_i,n}pte_\omega^\kappa$ *is a randomization of the transition probability, which is configured by a system variable* $\varphi_p \in [0 \dots 1]$.

$$ {}^{n_i,n}pte_\omega^\kappa = {}^{n_i,n}pt_\omega^\kappa \cdot (1 - \varphi_p) + \frac{\varphi_p}{\left|{}^n\mathcal{N}\right|} $$

Similar to φ_d, φ_p scales the influence of randomization continuously between the extremes of $\varphi_p = 0$, which bases the ants' decision on scents only, and $\varphi_p = 1$, which lets the ants pickup the next node uniformly at random.

3.6.7 Transition

At the beginning of each transition phase, the ant cleans its known type hierarchy by removing all types that do not belong to the type hierarchy of its primary resources ω_κ and have a similarity below a certain level σ towards these types.

$$ {}^a_{\tau+1}\mathcal{T} = \left\{ t \in {}^a_\tau\mathcal{T} \;\middle|\; \max_{t_i \in {}^a\mathcal{T}_{\omega_\kappa}} simd_t^{t_i} \geq \sigma \right\} \cup \bigcup_{\kappa \in \mathcal{K}} {}^a_\tau\mathcal{T}^*_{\omega_\kappa} $$

Subsequently, it updates the scent-list towards the chosen neighbor with typed pheromones. The pheromone update is modeled by adding 1 for each direct parent type of the ant's primary resource. In case they are not already present, these types, along with their super types, are added into the local type hierarchy of the current node.

$$ {}^{n}_{\tau+1}\mathcal{T} = {}^{n}_{\tau}\mathcal{T} \cup {}^{a}_{\tau}\mathcal{T}^{*}_{\omega_\kappa} \qquad {}^{n_i,n}_{\tau+2}sc^{\kappa}_{t_s} = {}^{n_i,n}_{\tau+1}sc^{\kappa}_{t_s} + \begin{cases} 1 & t_s \in {}^{a}\mathcal{T}_{\omega_\kappa} \\ 0 & \text{otherwise} \end{cases} $$

After that, the transition to the next node is made. For better visualization, this phase takes a randomized amount of time, where the ant is rendered at the current node link, simulating its movement. When the transition is complete, the ant re-enters the learning phase.

3.6.8 Returning

If an ant could successfully accomplish its task or if its time-to-live becomes zero, it switches into the return phase. As real ants might be a little more excited and hasty to return home in case they found something, successfully returning virtual ants run 20 percent faster and spread twice as much pheromones as regular ones. On the contrary, unsuccessful ants run 20 percent slower and do not drop any pheromones on their way back[4].

As semantic scents only lead to resources but not to their requestors, ants generally have no way of knowing the shortest path home. Yet, to establish short semantic trails, their path home should not be unnecessarily long. This extension provides the following two return strategies.

Taken Path Strategy

This is the default strategy, which suggests that ants return the previously taken path home, omitting possible loops for efficiency. It is supposed that, in the long run, short trails will emerge since ants use a randomized path selection probability. If, by chance, an ant discovers a shorter route to a resource, this route is enforced as ants taking this path will generally return faster, which causes the pheromone concentration to raise more rapidly and to maintain at a higher level. However, this effect is highly dependent on the optimal adjustment of the pheromone decay rates, the nature of the pheromone stimulus and the decision randomization. If the stimulus is too high, ants might easily reject shorter, yet later discovered routes. If it is too low, they might end up running randomly.

[4]The reason why they return at all is that nodes keep track of how many ants they produced for load balancing. In case no pheromones are spread, the decrease of the returning speed does not serve any technical purpose - its just meant as a way of visualizing demoralization.

Shortest Path Strategy

Therefore this second strategy is provided, which suggests that ants simply take the shortest path home. This strategy is non-distributed, and intended to be used only for a comparison with the first strategy during the system evaluation. It makes use of the NetLogo internals that allow the determination of the shortest path between any nodes from a system-global perspective. Nonetheless, a distributed version could be implemented later by an adaption of the ACO, which uses node scents.

3.6.9 Completion and Death

This phase is entered when an ant has returned home. It is meant to return possible user feedback (success / failure), to update statistics and node records, and to clear any references to the ant so that it can be removed in its final dying phase.

3.7 S-Box Clustering

The S-Box clustering is conducted by a combination of node processes and OUT-Ants. As described in the previous section, any node gains its partial ontological knowledge by its visiting OUT-Ants, as anytime they drop a triple, they add the type hierarchy of their resources to the local S-Box of the node. Besides, when updating the scent-lists, they also add this hierarchy, so that other visiting ants may understand how this new scent relates to the others.

To prevent the local knowledge from growing too large, which would compromise scalability, resources are continuously examined towards their local semantic suitability by a special background process. This process can create OUT-Ants for unsuitable resources, so that the corresponding triples are re-located. Analogously, another process periodically removes all entries below a certain local similarity from the local scent-lists. All types that are not refrenced by any resources or scent-list entries, are periodically removed by a third process, which guarantees the node S-Boxes to stay minimal. These and all other node processes are described in detail in section 3.9 (Node Processes).

As S-Box information is partially replicated among the nodes and ants, any S-Box insertions or deletions must be replicated as well. While insertions spread on-the-fly by new OUT-Ants, the only way to conduct deletions is to replicate them through the entire network and all active ants via a message cascade. Though this is very costly, it is reasonable to assume that S-Box deletions are generally very rare and the vast majority of S-Box operations are inserts or updates[5].

[5]Yet, updates are not part of the LINDA model and therefore not implemented in this thesis, but they can easily be realized as a minor modification of the RD-primitive.

3.8 Triple Retrieval

The semantic triple retrieval is conducted by a swarm of IN- or RD-Ants, which retain this information

- the assigned template ω
- a cluster indicator $\kappa \in \mathcal{K}$, indicating which of the template resources it is looking for (primary resource).
- a local type hierarchy ${}^{a}\mathcal{T}$, representing the ant's ontological understanding
- the similarity distribution ${}^{a}simd$ for the types of its hierarchy
- its home-node ${}^{a}n$, its time-to-live *ttl*, and a path-memory
- three additional path-memories for locking
- a request-id

In addition to the previous URI-Cluster implementation by Anne Augustin, a semantic cluster allows support for typed triple retrieval. To distinguish a plain template entry from a typed request, a special URI-scheme is introduced, where, to mark a template entry as type-request, the type's URI is used as scheme-specific part and the namespace *rdf-typed* is preceded. So, to find any resource of a fictional type *ns://StemBolt*, the template entry would be *rdf-typed://ns://StemBolt*. Irrespective of whether a template contains plain or typed entries, its matches are found concurrently by IN- or RD-Ants following the types of their primary resource.

3.8.1 Concurrent Matching

For each template which gets inserted into the Semantic Cluster, three IN- or RD-Ants are created - one for each template resource. Furthermore, a Universally Unique Identifier (UUID) is created for that request and assigned to the ants in their initialization phase. This id is also maintained at the current node, to keep record of the incoming results. Only the first result is returned to the requestor and, in case of an IN-request, only this result is removed from the Triple Space, ignoring the results of subsequently returning ants.

Unless the resource equals a wildcard, the ants follow the types of their resource to find a template match, in other case they perform a random walk. On their way, IN- and RD-Ants run basically through the same phases like OUT-Ants, yet their differ at the task fulfillment and completion phase.

3.8.2 Task Fulfillment

The task fulfillment phase of IN- and RD-Ants is completely different from that of OUT-Ants, as they do not drop a triple, but search for one to match their template. This match-making, plain or typed, is conducted by looking through the local set of triples at the current node. In the process, every ant evaluates these triples to fully match their template, not just their primary resource. Since the implemented A-Box data structure indexes triples by their primary resource and the S-Box implementation indexes resources by their direct parent types, the set of possible matchings can be limited to a great deal. This prevents the ant from having to look through all present triples, which would be very costly. In case the template resource is plain, this set is restricted to just the statements which have the desired resource as primary resource. In case the template entry represents a type, it is restricted to those triples which have the desired type or one of its subtypes as direct parent type of their primary resource.

If the ant's primary resource is a wildcard, it apparently must consider all local statements. Therefore, the creation of wildcard ants is omitted as long as the template contains plain or typed entries. As non-wildcard ants have a type information to follow and a resource to limit the search space, they can find matches faster and more efficiently. In fact, wildcard ants are only created when the template consists only of wildcards, and then just one subject IN- or RD-Ant is created. As any statement matches in this case, these ants can find their result quickly and will not comprise the system performance.

To increase performance at another spot, IN- and RD-Ants which by chance come along their home node, look up the local record for their request in their learning phase, to find out if the request has already been fulfilled by another ant. If this is true, they switch to the completion phase immediately.

While RD-Ants can simply carry home a copy of a found match, an IN-primitive requires the found match to be removed and to be returned only once. To ensure this, IN-Ants have to apply a concurrent and distributed locking strategy, which is described next.

3.8.3 Distributed Locking

Anytime an IN-Ant finds a match, it must ensure that it returned and removed only once. Therefore, it creates three locking ants - one for each resources of the match. These ants behave like RD-Ants themselves, except that their only task is to atomically lock the match in their cluster incarnation. Since the locking ant that is created for the cluster incarnation of its parent, can find its triple at the very same node, it does not have to traverse the network. Hence, for efficiency reasons, the parent IN-Ant simply locks the match directly and creates locking ants only for the other two incarnations. *These* ants generally have to conduct a search to find the duplicates of the match.

In contrast to RD-Ants, locking ants return the request-id of the IN-Ant that has acquired the triple lock. This might be the request-id of the IN-Ant which created them - in case the triple has not been locked before - or the request-id of any other IN-Ant that is concurrently trying to lock the triple. If a locking ant cannot find the triple in the cluster within its time-to-live, because the triple might have been removed concurrently, it simply returns *nil*.

When all locking ants have returned, the competing IN-Ants have to make a consistent and distributed decision of who gets the permission to retrieve and remove the triple. Unlike in the cluster implementation of Anne Augustin, where locking attempts were randomly repeated until an ant could acquire the lock in all three cluster incarnations, this thesis implements a strategy to consistently grant the remove permission instantly right after the first attempt. In doing so, the common agreement is established, that this permission is always granted to the IN-Ant that acquires the lock in the subject cluster. This guarantees that at most one IN-Ant gets the triple access, since the subject lock can only be acquired once and loosing IN-Ants dismiss the match and continue their search. If an IN-Ant gets the triple access, it returns home carrying a copy of the match.

This strategy bases on the invariant of the LINDA model that all triples exist exactly once. If this invariant is violated, triple deletions could cause inconsistencies between the cluster incarnations. If, for instance, a triple exists multiple times in one incarnation and only once in another, a deletion of that triple could leave stale copies in the cluster. The original LINDA model describes a non-distributed associative memory. There, subsequent insertions of existing triples have no effect as similar triples are mapped to the same memory address. In a distributed space, it is not possible to *instantly* merge similar triples as their is no global memory and these triples might be inserted at completely different locations. Yet, if similar triples cannot be merged *instantly*, the invariant is violated. For this reason it is inescapable to base deletions on triple identity. This identity is created by assigning a unique id to each triple and its replicas, which is a technique that is also used in some commercial triple space implementations like TSpaces or GigaSpaces (see section 2.6 (Related Work), page 36). As a result, *identical* triples exist only once in each incarnation, but *similar* triples may exist multiple times. In this respect, the implementation deviates from the original LINDA model.

3.8.4 Completion

After they come back, IN- and RD-Ants check the record for their request at their home node. If the ant turns out to be the first, it returns the match and, in case of an IN-Ant, removes the triples and its locks in all three cluster incarnations. Therefore it keeps a memory of the paths of its locking ants as well. Later arriving ants for that request simply terminate their (meaningless) existence. Yet, in case of IN-Ants that found a different match, they remove their locks first without removing the triple itself.

3.9 Node Processes

3.9.1 Triple and Template Processing

As mentioned earlier, every node contains a pending lists for new triples and templates. Any entries are removed from these queue by this process as long as the number of ants stays below the configurable maximum ant count. Then the process creates the appropriate IN-, OUT- and RD-Ants for each removed item.

3.9.2 Cluster Maintenance

This process continuously examines the local resources to remove the most dissimilar ones.

Definition 3.10 (Most Dissimilar Resources) *The set of* most dissimilar resources ${}^{n}\mathcal{R}_{\diamond}^{\kappa}$ *at a node* $n \in \mathcal{N}$ *for a cluster indicator* $\kappa \in \mathcal{K}$ *is defined by all resources* $r \in {}^{n}\mathcal{R}^{\kappa}$*, with a local similarity* ${}^{n}lsim_{r}^{\kappa}$*, that is below the node local similarity* ${}^{n}\varsigma^{\kappa}$ *multiplied by the minimum similarity bound msb (system variable).*

$$ {}^{n}\mathcal{R}_{\diamond}^{\kappa} = \{r \in {}^{n}\mathcal{R}^{\kappa} \mid {}^{n}lsim_{r}^{\kappa} \times \ msb < {}^{n}\varsigma^{\kappa}\} $$

Definition 3.11 (Local Similarity) *The* local similarity ${}^{n}lsim_{r}^{\kappa}$ *of a resource* $r \in {}^{n}\mathcal{R}^{\kappa}$ *at a node* $n \in \mathcal{N}$ *and for a cluster indicator* $\kappa \in \mathcal{K}$ *is defined as the average local similarity of its direct parent types. The local similarity of a type is the sum of all similarity-weighted values in the similarity distribution at that node, divided by the local statement count.*

$$ {}^{n}lsim_{r}^{\kappa} = \frac{\sum\limits_{t \in {}^{n}\mathcal{T}_{r}} \sum\limits_{t_i \in {}^{n}\mathcal{T}} {}^{n}td_{t_i}^{\kappa} \cdot simd_{t}^{t_i}}{\left|{}^{n}\mathcal{T}_{r}\right| \cdot \left|{}^{n}\Omega^{\kappa}\right|} \quad , \quad \left|{}^{n}\Omega^{\kappa}\right| \neq 0 $$

Definition 3.12 (Node Local Similarity) *The* node local similarity ${}^{n}\varsigma^{\kappa}$ *of a node* $n \in \mathcal{N}$ *and a cluster index* $\kappa \in \mathcal{K}$ *is the average triple-count-weighted pairwise similarity of all types in the local triple distribution* ${}^{n}\Omega^{\kappa}$*.*

$$ {}^{n}\varsigma^{\kappa} = \frac{\sum\limits_{t_i \in {}^{n}\Omega^{\kappa}} \sum\limits_{t_j \in {}^{n}\Omega^{\kappa}} {}^{n}\Omega_{t_i}^{\kappa} \cdot {}^{n}\Omega_{t_j}^{\kappa} \cdot sim(t_i, t_j)}{|{}^{n}\Omega^{\kappa}|^2} $$

The maintenance process picks up one of the most dissimilar resources r, removes all unlocked triples $\omega \in {}^{n}\Omega^{\kappa}$ that have r as primary resource and assigns them to OUT-Ants to relocate them.

$$ {}^{n}_{\tau+1}\Omega^{\kappa} = {}^{n}_{\tau}\Omega^{\kappa} \setminus \{\omega \in {}^{n}_{\tau}\Omega^{\kappa} \mid \omega_{\kappa} = r\} $$

3.9.3 Pheromone Decrease

At the end of each time step, this process decreases the entries of all scent-lists by the *pheromone decay rate* $\theta \in [0 \ldots 1]$, simulating the natural evaporation of scents. Besides, any pheromones below the minimum pheromone level ψ are automatically removed from the lists.

$$ {}^{n_i,n}_{\tau+1}sc^{\kappa}_{t} = \begin{cases} {}^{n_i,n}_{\tau}sc^{\kappa}_{t} \cdot \theta & {}^{n_i,n}_{\tau}sc^{\kappa}_{t} \geq \psi \\ nil & \text{otherwise} \end{cases} $$

3.9.4 Template Generation

As described in section 3.2.4 (Triple Space Architecture), page 43, this process creates random templates for present triples and assigns them to special RD-Ants. These ants first perform a random walk and then start the retrieval. As the generated ants spread their pheromones just like regular ants, they keep the cluster trails up-to-date and prevent the semantic trails from disappearing completely, even if there are no active external requests. In the simulator, the *template generation rate* for all nodes is globally configured[6].

3.9.5 Garbage Collection

To ensure that the S-Boxes of the nodes stay minimal, this process continuously updates the local S-Boxes by removing any class or property definitions that are no longer related to any local resources, due to triple operations like deletions or relocations, or because of scent-list updates.

$$ {}^{n}_{\tau+1}\mathcal{T} = {}^{n}_{\tau}\mathcal{T} \setminus \{t \in {}^{n}_{\tau}\mathcal{T} \mid \forall \kappa \in \mathcal{K} : \nexists \omega \in {}^{n}_{\tau}\Omega^{\kappa} : t \in {}^{n}_{\tau}\mathcal{T}^{*}_{\omega_\kappa} \wedge t \notin {}^{n,n}sc^{\kappa}\} $$

This concludes the theoretical semantic clustering and retrieval approaches. The next chapter describes their implementation and the assembled simulator software.

[6] A possible improvement to a global configuration could be self-adjusting template generation rate, depending on the actual scent level. Nonetheless, this is not implemented as a part of this thesis

Chapter 4

Implementation

In this chapter, the installation and build of the Triple Space implementation is described. It includes a complete manual of the simulator user interface and its functionalities and additionally explains the most crucial implementation details.

4.1 Installation

4.1.1 Requirements

Java

The Semantic Cluster implementation consists of the NetLogo application framework, a NetLogo model which is programmed in the NetLogo modeling language, and a variety of NetLogo extensions written in the Java Programming Language. Thus, a minimal installation of the Java Runtime Environment (JRE) is required to run the program. Yet, it is recommended to use a Java Development Kit (JDK) installation, which provides the *-server* option for better performance.

The extensions were implemented and tested under the 32-bit JDK version 1.6.0_11 and the 64-bit JDK version 1.6.0. The 1.6 versions of the JDK or JRE for Windows, Solaris and Linux are available at the official Java website by SUN Microsystems[50]. The JDK version 1.6 for MacOS X 10.5 can be obtained at the Apple update site[1].

Eclipse

Eclipse is an Integrated Development Environment (IDE) used in the implementation process. It can be downloaded from the Eclipse Foundation website [51] (version 3.3 or above is recommended).

[1] http://www.apple.com/support/downloads/javaformacosx105update1.html

The Eclipse Projects, which contain the models and sources, are available on the appended CD. They can also be downloaded from the Subversion Repository of the *AG Netzbasierte Informationssysteme* of the *Freie Universität Berlin*[2]. If the Eclipse IDE is used, it is recommended to use its *Subversive* plugin as a subversion integration[3]. Alternatively, arbitrary stand-alone subversion clients can be used to download the projects directly to the local file system.

Ant

Apache Ant is a Java-based build tool that can be used to compile and assemble the simulator. Though Ant is not essentially required, the projects contain certain Ant scripts that simplify the build of the system. The Eclipse IDE is bundled with a complete Ant distribution, hence there is no need for a separate Ant download when using Eclipse. Otherwise, it can be downloaded from the Apache Ant website[4] and extracted into a directory of choice. In order to execute the provided scripts, this directory must be included in the default search path of the operating system.

Maven

Apache Maven is a sophisticated build management tool for Java projects, which is required for a build of the system from scratch . Properly configured, Maven automatically compiles the sources, creates the appropriate JAR files, includes the necessary NetLogo manifest entries, obtains all referenced libraries and creates the API documentation. Maven requires a description of each project in the form of an XML document called Project Object Model (POM). After downloading Maven from the Apache Maven website[5], it is installed by extracting it into an arbitrary directory and adding M2_HOME/*bin* to the default search path of the operating system.

4.1.2 Software Overview

Projects

The simulator consists of five different projects:

SwarmLindaExtensions contains the previous Java-Extensions by Daniel Graff, which basically include numeric utility classes like random number generators.

RDFExtensions provides shared classes with general RDF-related functionality (e.g. importing ontologies from files, utility classes for using the Jena API or data

[2]Contact information is available at `http://www.ag-nbi.de`.
[3]Download and installation instructions available at `http://www.eclipse.org/subversive/`.
[4]`http://ant.apache.org/`
[5]`http://maven.apache.org/download.html`

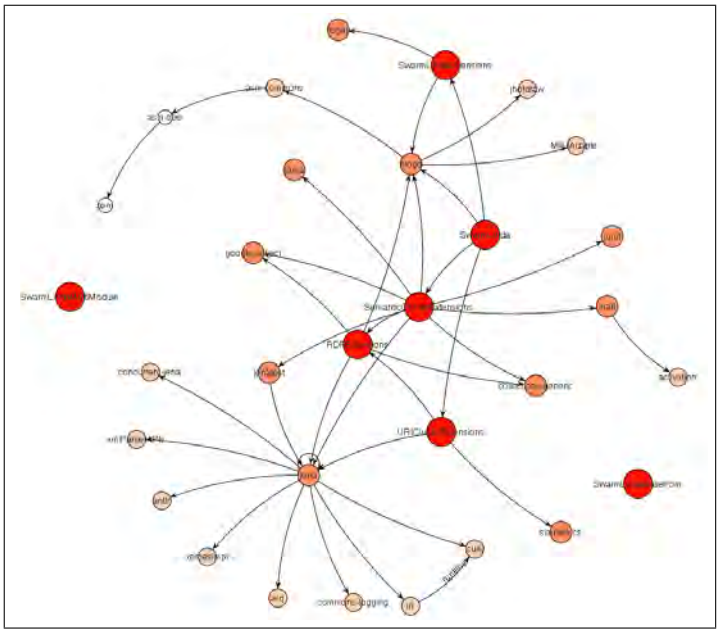


Figure 4.1: Dependencies of the simulator projects

structures to store and index triples by their primary resources). It is used by both, the *URIClusterExtensions* and the *SemanticClusterExtensions* project. Besides Jena, it utilizes the generic version of the Apache Collections Framework and the Google Collection Library.

URIClusterExtensions gathers all Java sources of the URI-Cluster extensions implemented by Anne Augustin. It imports the SimMetrics library, which includes implementations for string edit distance calculations.

SemanticClusterExtensions contains the extensions of this thesis. It uses the Apache Collections Framework, the Google Collection Library, the Jena API, and the Jama library.

SwarmLinda is the main project. It includes build scripts, various POMs and the overall NetLogo model, which contains the code written in the NetLogo programming language and calls the Java extensions. The model itself is divided into several files - the network simulator (*network-generation.nlogo*), the main model (*swarm-linda-simulator.nlogo*), the URI-Cluster model (*URI-C.nls*), and the Semantic Cluster model (*SEM-C.nls*). The compiled system is assembled in the *build* subdirectory of this project.

A tree representation of the relations and dependencies of the projects is shown in figure 4.1. It is derived by the Maven Overview Plugin[6]. A brief explanation of the used external libraries is given next.

[6] http://code.google.com/p/maven-overview-plugin/

Libraries

NetlogoApp provides all NetLogo Java classes and allows to start the simulator as a stand-alone application. Through this library, it is possible to access the state of the simulator and certain NetLogo internal functionalities, like agent creation or shortest path determination, from within the Java extensions.

Log4J The Apache Log4J library contains a sophisticated Java logging facility. It is used by all extensions.

Apache Collections Framework and the Google Collection Library The Google Collections Library and the *collections15* project from Larvalabs provide a variety of multi-purpose data structures additional to the *java.util.** and *java.concurrent.** classes.

Jena Semantic Web Framework provides support for processing RDF, RDF-S and OWL ontologies. Furthermore, it contains a SPARQL search engine as well as several rule-based reasoner implementations. It is developed by the Hewlett Packard Development Company, LP.

SimMetrics is an open source library developed by the Sheffield University, which provides string comparison algorithms, such as the *Levenshtein Edit Distance* or *Cosine Similarity*.

Jama is a basic linear algebra package for Java that includes a facility for solving linear equations. It is provided by the Mathematical and Computational Sciences Division of the National Institute of Standards and Technology (NIST), Gaithersburg, USA.

4.1.3 Simulator Build and Launch

Each project can be built by executing the *Maven install command* (*mvn install*) in its base directory. The SwarmLinda project contains an additional POM, defining a multi-module project that envelops all other projects. It allows a combined build, which is triggered by executing the Maven install command from *SwarmLinda/poms/multimodule*. During the execution, all sources are compiled and bundled into JAR-files. For each JAR, a manifest file is created that designates the extension class manager, which is required by NetLogo for proper initialization. Thereafter, these bundles are copied to the output directory (*SwarmLinda/build*) as are the libraries they depend on. The required external libraries are, if not already present in the local Maven repository, automatically retrieved from remote repositories defined in the project descriptions. At last, the API documentation of all projects is generated and copied into the output directory as well. The definitions and configurations of all projects derive from a single base POM, located at *SwarmLinda/poms/multimodule/base-pom*.

When the build is completed, the simulator can be started by executing the following command from the *SwarmLinda/build* directory:

java -jar Main/SwarmLinda-1.0-SNAPSHOT.jar -Dfile.encoding=utf8.

To improve performance, it is recommended to also apply these additional VM options:

-Dsun.java2d.noddraw=true -Xmx1024m -XX:PermSize=32m -XX:MaxPermSize=128m -server.

There are several batch files included in the build directory to start the program under Windows. They can be examined for further options and adapted for other operating systems.

In order to configure the projects to work in Eclipse, an additional Eclipse classpath variable *M2_REPO* has to be defined, pointing at your local Maven repository (which is USER_HOME/*.m2/repository*, unless manually configured otherwise). The variable is added under *Windows/Preferences/Java/Build Path/Classpath Variables*. The sources of the external libraries can be downloaded and attached to the classpaths of the projects by running the Maven Eclipse Plugin (*mvn eclipse:eclipse*) in the base directory of the multi-module project.

4.2 User Manual

4.2.1 Network Creation

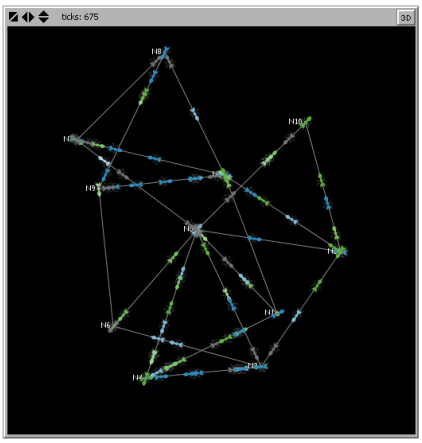
Figure 4.2: The Network Generator interface

The Network Generator was originally implemented by Daniel Graff as a part of SwarmLinda. Besides being ported to the latest NetLogo version 4.0.2, which includes some changes in the modeling language, it was basically left unchanged and can be used to create networks for the Semantic Cluster. Its interface (see figure 4.2) is quite self-explanatory. It basically allows the creation and deletion of nodes and links and layouts them appropriately. A complete description of the generator is given in the work of Graff [2].

4.2.2 Simulator

The simulator interface consists of the Main View and a variety of sections which allow to configure, control, monitor, evaluate and interact with the system. Figure 4.3 shows a screenshot of the complete interface. Its functionalities are described in the next sections.

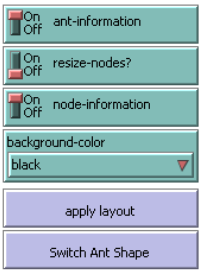
Main View



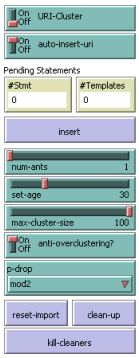
The *Main View* displays the cluster nodes and their links. It furthermore shows the locations of all active ants. The ant coloring indicates its kind – OUT-Ants are colored blue, IN-Ants are green and returning ants appear grey. The different shades of the coloring represent the ants' cluster incarnations. NetLogo provides scaling and rotating functionality as well as a context menu that allows the selection and inspection of single nodes and ants. A very useful feature is the "follow agent" functionality which eases isolated ant observations.

Layout Settings

The *Layout Section* allows to display additional ant and node information in the Main View, such as the ants' time-to-live or the number of triples located at the different nodes. It furthermore contains features to resize the nodes according to their degree and to define the background color of the Main View. It is also possible to assign one of three different shapes to the semantic ants.



URI-Cluster Configuration



This section contains the controls of the URI-Cluster. Both, the URI-Cluster and the Semantic Cluster, can be switched on and off independently. As they were developed in parallel, the URI-Cluster implementation was not fully completed at the time of the simulator development. Therefore, in order to conduct the type identification, the Semantic Cluster simulates the results of the URI-Cluster, by assembling the type hierarchy of the inserted triples and templates accordingly. The conceptual approach for the integration of both clusters is described in section 3.3 (Type Identification). At this point, the URI-Cluster has to be turned off to let the Semantic Cluster work correctly.

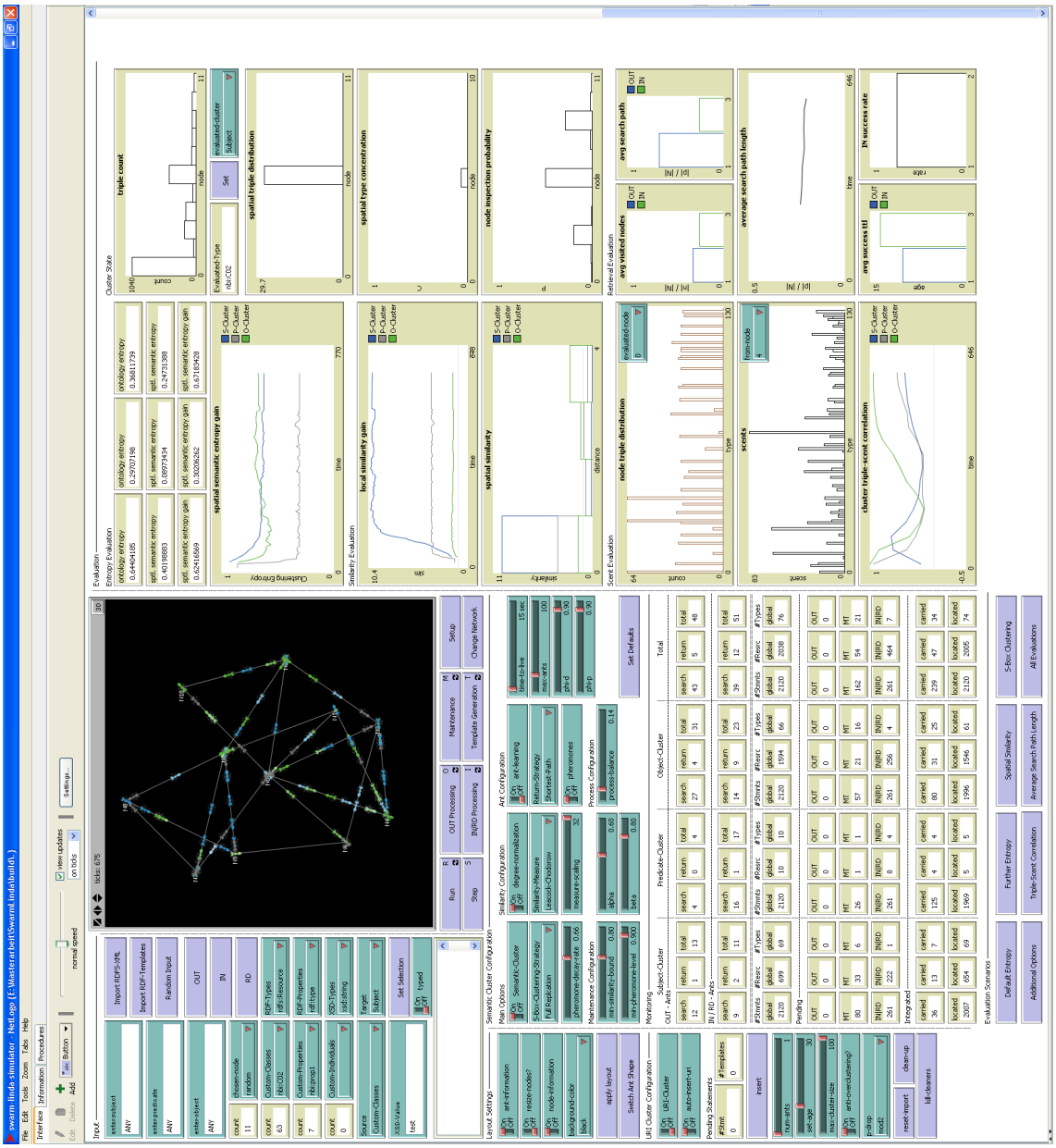
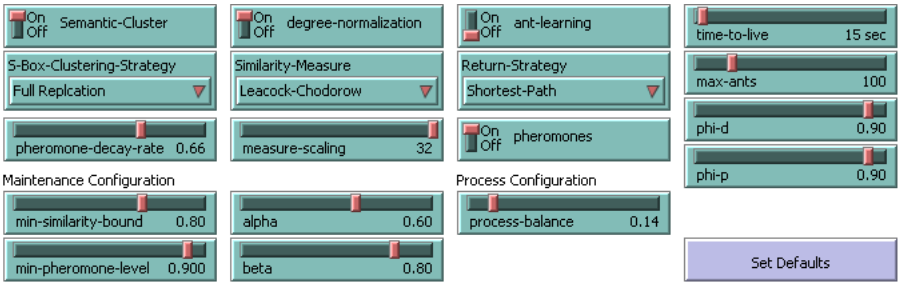


Figure 4.3: The simulator main view

Semantic Cluster Configuration



The Semantic Cluster provides these configuration options:

S-Box Clustering Strategy allows to select one of three S-Box clustering strategies. If the *Full Clustering* is used, ants and nodes maintain their own S-Boxes as described in section 3.2.2 (T-Box Clustering), page 41, and perform all similarity calculations themselves. If the *Full Replication* strategy is chosen, the ants use the global S-Box, which is maintained for evaluation purposes. The *Experimental* strategy is a modification of the *Full Clustering*. It likewise distributes the S-Box, but lets the nodes run the similarity calculation periodically. The results are then used by all visiting ants.

Pheromone Decay Rate controls the amount of pheromone decay per minute.

Minimum Similarity Bound defines the lower bound for type similarity in the node-local S-Boxes. Types with a lower local similarity than *the minimum similarity bound* $\times$ *the average node local similarity* are relocated by the maintenance process.

Minimum Pheromone Level defines the lower bound for pheromones within the scent-lists. Entries below that boundary are removed instantly.

Degree Normalization turns the degree normalization on or off.

Similarity Measure allows to choose the underlying similarity measure.

Measure Scaling, Alpha, Beta adjust the scaling variables for the chosen similarity measure. The *measure scaling* is used as γ or δ, depending on the selected measure, *alpha* and *beta* apply only to Haase-Siebers-Harmelen (see section 3.4.4 (Measure Scaling), page 51).

Ant Learning is only in effect if the S-Box clustering is activated, in which case it controls, whether the ants learn from their visited nodes only (off) or other present ants as well (on). If activated, the ants gain their knowledge more quickly, but the learning phase becomes more costly, causing a general performance decrease.

Return Strategy provides the options *Taken Path* and *Shortest Path*.

Pheromones enables or disables the pheromone dropping of the ants.

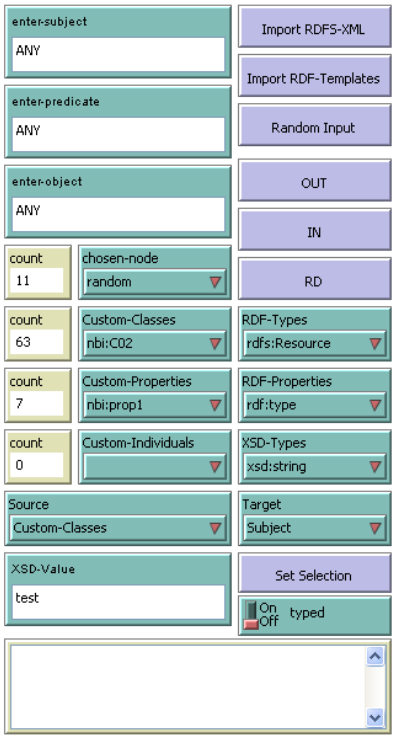
Process Balance Rate continuously adjusts the priority between the maintenance process (left) and the template generation process (right).

Time-To-Live defines the initial time-to-live of all ants.

Maximum Ant Count the maximum number of ants in the Semantic Cluster.

Phi-D, Phi-P adjust the randomization of the ant decisions of whether to drop a triple (*Phi-D*) or which node to visit next (*Phi-P*).

Input Area



The input area provides miscellaneous functionalities for triple insertion and retrieval. The triples (or templates) can be defined using the interface, or be loaded from file. To import an RDF/XML ontology, the file is chosen in the file dialog which is opened by the *Import RDF/XML* button. During the import, the XML ontology is decomposed into RDF-triples, which are stored in the pending queue of the URI-Cluster.

Unless the *auto-insert* option is activated, the triples must be manually entered into the URI-Cluster using the *Insert* button. The URI-Cluster then stores the S-Box triples for type identification and routes all triples to the Semantic Cluster. There, the types of the resources are identified and the appropriate OUT-Ants are created. Furthermore, it is possible to import triple templates from a file using the *Import Templates* button. These templates are analogously processed and routed to the Semantic Cluster.

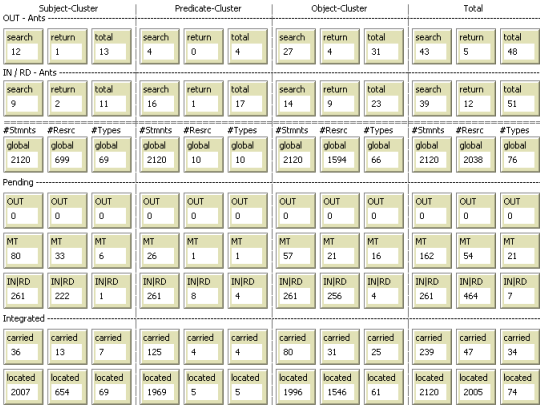
Manual primitive executions are conducted using the three textfields to enter custom subjects, predicates, and objects, and the primitive buttons *IN*, *OUT* and *RD*. The executing node can be chosen specifically or at random in the *chose-node* combobox. To ease manual triple definement, the input area provides various choosers containing already present types, properties, and instances. The *Set Selection* button is used to assign any of these predefined resources to a triple field, where the source chooser and the target triple field are specified using the *Source* and the *Target* widget. Besides, it is possible to define XSD-resources by selecting the XSD type in the *XSD-type* chooser and by entering the string representation of the value in the *XSD-value* field. The *Random* button is used to pick up a random triple in the cluster and generate a template (plain or typed). Finally,

the results of the primitive executions and occurring time-outs are shown in the output area at the bottom.

Cluster Control

This section controls the different activities of the clusters. The *Run* and *Step* buttons activate the progression of time by one tick (*Step*) or continuously (*Run*). Furthermore, the ant creation can be triggered independently for each ant kind (OUT and IN/RD). The template generation and maintenance process can be activated separately using the *Template Creation* and *Maintenance* buttons. A complete reset of the cluster is initiated using the *Setup* button. The *Change Network* button opens a dialog to load a different network from a file.

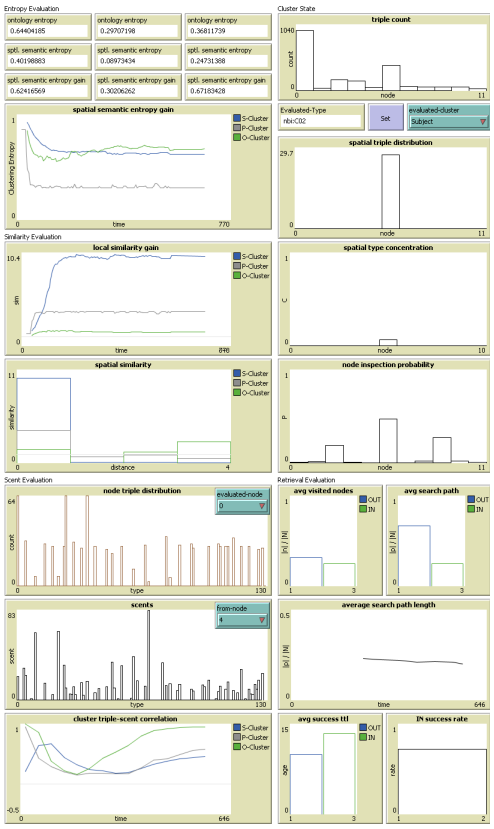
Monitoring



The monitoring section is continuously updated by a background thread to provide a detailed insight of the current system state. It is divided into four major columns - one for each cluster incarnation and one for a total view - with three subcolumns each. While the first two lines provide information about the active ants and their activities, the following indicate the state of the cluster.

In each major column, the first line shows the number of searching, returning and total OUT-Ants (including maintenance), while the second line provides these figures regarding to IN- and RD-Ants. In the cluster segment, each first subcolumn indicates the total number of A-Box-Triples in the ontology (line 3), the number of pending OUT triples (line 4), the number of pending maintenance triples (line 5), and the number of pending templates (line 6). Furthermore, the total numbers of currently carried and located triples are given in line 7 and 8. For each of these monitored triple fractions, the second subcolumn displays the number of distinct primary resources of these triples, and the third subcolumn shows the number of the distinct direct parent types of their primary resources.

Evaluation



This part of the interface displays the results of the cluster evaluations. The most important quality indicator is the spatial sematic entropy gain, which is displayed in the *Entropy Evaluation* area. There, the current values are given for each cluster incarnation, showing the ontology immanent entropy (first line), the spatial sematic entropy (second line) and their quotient, the spatial sematic entropy gain (third line). The chart below displays the progression of the entropy gain for all incarnations.

In the *Similarity Evaluation* area, the progression of the node local similarity gain is displayed, which is the average similarity of co-located triples, relative to the average triple similarity of the un-clustered ontology. The spatial similarity evaluates the average similarity of triples against their network distance to verify the formation of semantic neighborhoods.

The *Cluster State* section provides a more detailed insight of the current distribution of triples. In the upmost chart, it shows the distribution of *all* triples in the selected cluster incarnation. The resource that is currently selected in the *Input Area* can be chosen for the type specific evaluations below, using the *Set* button. There, the first diagram shows the spacial triple distribution for the selected type in the selected cluster incarnation. The concentration of these triple is displayed in the *Spatial Type Concentration* chart below. Based on the current semantic trails and the current value of *Phi-P*, the *Node Inspection Probability* for the selected type is calculated and shown in the corresponding chart at the bottom of this area.

The node inspection probability is derived from the node transition probabilities of a potential OUT-Ant carrying a triple of the selected type. These transition probabilities are used to determine the *stationary distribution of a Markov chain* for the cluster network, which is then displayed in the chart.

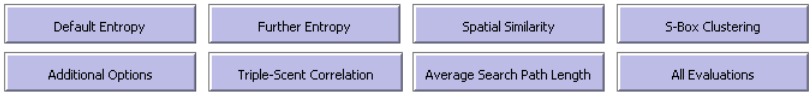
Markov chains model probabilistic walks in general graphs[7]. The model consists of a vector π_τ which, for each node, contains the probability of being there at the time τ, and the matrix M, which contains the node transition probabilities. The progression of one time step corresponds to the multiplication $\pi_{\tau+1} = M \times \pi_\tau$. As shown in [52], if all nodes

[7]Originally, Markov chains were used to model probabilistic transitions in state machines, which were represented as graphs.

are connected and all transition probabilities are greater than zero, a stable probability distribution (stationary distribution) $\bar{\pi}$ arises, independent of the initial distribution π_0. This stationary distribution can be determined by solving the linear equations of $\bar{\pi} = M \times \bar{\pi}$. Using the JAMA library, the stationary distribution for the potential OUT-Ant is calculated and displayed in the *Node Inspection Probability* diagram.

The quality of the sematic trails is investigated in the *Scent Evaluation* area. There, two choosers allow to select a reference node - the triple distribution of which is shown in the upper diagram - and one of its neighbors - the outgoing scent-list of which towards the reference node is displayed in the diagram below. In both diagrams, the type-entries of the distributions are consistently sorted left to right by their decreasing similarity towards the type with the highest local similarity at the reference node. The bottommost chart shows the average correlation between the triple distributions and the incoming scent-lists in the cluster. This correlation indicates the quality of the semantic trails, since it quantifies their appropriateness in the path selection process.

The triple retrieval is analyzed in the *Retrieval Evaluation* area, which contains five charts. The upper left chart displays the average number of nodes visited by OUT-, IN-, and RD-Ants, not counting repeated visits. In contrast, the upper right chart shows the average search path length, which counts repeated visits. In both charts, these figures span the time of the last 60 seconds and they are given relative to the network size. The progression of the average search path length is displayed in the center chart. The average time-to-live left at the moment of the ants' deaths and the total IN-Ant success rate are displayed in the two diagrams at the bottom.



Finally, the *Evaluation Scenarios* section allows to run automated tests, which are used for the system evaluations. Their results and the complete descriptions of all calculations are given in chapter 5 (Evaluation).

4.3 Implementation Notes

4.3.1 Concurrency in NetLogo

The NetLogo programming language allows to define global, environment- and agent-specific variables and instructions. By default, the instructions are executed sequentially as a block for one agent at a time. This eases the implementation of simple models and agent behavior without the need to consider synchronization issues. If the execution time of these blocks is long, a sequential execution can cause an unrealistical simulation of activities, since meanwhile all other agents are blocked. Therefore, NetLogo allows instructions to be processed in an intermixing mode. Although the language keyword *ask-concurrent* implies otherwise, this does not mean that they run asynchronously. Instead, the included block is executed one instruction per agent, until all blocks are completed. Hence, also in this mode the agent activities are processed sequentially with mutual exclusion. In case an instruction calls a Java extension that is long running (e.g. a complex calculation), all other agents are stalled as well and have to wait until it is finished.

4.3.2 Concurrency in the Extensions

In a strict sense, both, the sequential and the intermixed code execution mode, are inappropriate for a lifelike simulation of a swarm-based system. In a real Triple Space, concurrent code execution is immanent as agents are created and disposed continuously. The corresponding processes must be autonomous and independent of the execution time, the data accesses, or possible runtime failures of other agents. Moreover, an asynchronous code execution allows to run calculations concurrently on multiple processors, if present. In contrast, NetLogo executes all instructions in a single worker thread.

For these reasons, this thesis evades the NetLogo execution modes and, despite the increase of complexity, implements a real multi-threaded system and the corresponding data access synchronization. Algorithms and shared data structures have therefore been designed in a thread-safe way, with minimal mutual exclusion (mainly by using thread-safe collections from the *java.util.concurrent* and Google Collections API, and by using reentrant read- and write-locks for additional synchronization). The most important data structures and their internal synchronization strategies are explained in the next section.

4.3.3 Main Data Structures

ABox

Each node contains an ABox for each cluster incarnation. Besides its set of triples, an ABox also contains two indices. The first is used to index all triples by their primary resource,

while the second indexes them by their non-primary (secondary) resources. This allows a fast triple matching for a given resource. Additionally, ABoxes keep records of all triple locks. Listings 4.1 and 4.2 give a code extract of the ABox fields and their initialization.

For maximum throughput, all ABox read operations (like *size*, *get_lock_id_for_a_triple* or *get_triples_for_a_certain_resource*) are thread-safe without synchronization, due to the underlying collections. As returned collections of triples or resources may change concurrently, all algorithms are implemented to handle such asynchronous modifications. Either they work correctly with any collection state or, in some cases, they create a thread-local defensive copy first.

All ABox write operations (*add_triple* and *lock_triple*) are thread-safe as well and execute concurrently (in the case of locking, only one thread gets the lock, of course). Remove operations are more complex, since they must be atomic to be consistent. They are therefore mutually excluded against other remove or write operations for the same triple. This exclusion is synchronized internally by locking the triple to define the critical scopes (see Listings 4.3, 4.4 and 4.5). As concurrent operations for the same triple occur rarely, all ABox operation run without any mutual exclusion most of the time.

S-Box

S-Boxes contain the set of their known types and three indices. One index is used to store direct super-types, one is used to store direct sub-types, and the third index stores the types of each present non-type resource. The latter index allows a fast conduction of typed matching. The basic RDF types and their relations are, by default, contained in all S-Boxes. Listing 4.6 shows the declaration of the fields and the used collection implementations.

Anytime a triple is added to an SBox, it is determined whether it is a class-, property-, subclass-, subproperty- or individual type definition and added to the indices accordingly. As direct super-types may become indirect super-types due to later triple additions, the indices are cleaned afterwards. This cleaning is conducted lazily when needed, since it is rather costly and not necessary within bulk operations like multiple triple insertions or SBox merging. Like ABoxes, SBoxes also provide concurrent read and insert operations, while triple deletions are mutually excluded against other delete or write operations. Listings 4.7, 4.8 and 4.9 show some excerpts of this mutual exclusion.

Since they require the determination of the height and distance, the LCS, and the MIS of the involved types, similarity calculations are very costly operations. Hence, they are always performed in background and, in case of the maintenance process, a defensive copy is used to allow parallel updates of the original SBox.

Listing 4.1: ABox Fields (ABox.java)

```
private final Field primaryResourceField;
private final Set<Triple> triples = new ConcurrentHashSet<Triple>();
private final ConcurrentSetMap<Node, Triple> primaryIndex = new ConcurrentSetMap<
    Node, Triple>();
private final ConcurrentSetMap<Node, Triple> secondaryIndex = new ConcurrentSetMap<
    Node, Triple>();
private final ConcurrentMap<Triple, UUID> lockedTriples = new ConcurrentHashMap<
    Triple, UUID>();
private final ConcurrentWeakLockStore<Triple> lockStore = new
    ConcurrentWeakLockStore<Triple>();
```

Listing 4.2: ABox Initialization (ABox.java)

```
public ABox(Field primaryResourceField) {
    this.primaryResourceField = primaryResourceField;
}
```

Listing 4.3: ABox Triple Locking (ABox.java)

```
public boolean lockTriple(Triple match, UUID requestId) {
    lockStore.lock(match);
    try {
        if (!triples.contains(match))
            return false;
        UUID oldLock = lockedTriples.putIfAbsent(match, requestId);
        return oldLock == null || oldLock.equals(requestId);
    }
    finally {
        lockStore.unlock(match);
    }
}
```

Listing 4.4: ABox Conditional Triple Removal (ABox.java)

```
private boolean removeTripleIfNotLocked(final Triple triple) {
    lockStore.lock(triple);
    try {
        return getLockId(triple) == null && removeTripleAndLock(triple);
    }
    finally {
        lockStore.unlock(triple);
    }
}
```

Listing 4.5: ABox Unconditional Triple Removal (ABox.java)

```
public boolean removeTripleAndLock(Triple triple) {
    if (!triples.remove(triple))
        return false;
    for (Field field : TRIPLEFIELDS) {
        Node resource = field.getField(triple);
        getResourceIndex(field).remove(resource, triple);
    }
    removeLock(triple);
    return true;
}
```

Listing 4.6: SBox Fields (SBox.java)

```
private final ConcurrentSetMap<Node, Node> directSubTypes = new ConcurrentSetMap<
    Node, Node>();
private final ConcurrentSetMap<Node, Node> directSuperTypes = new ConcurrentSetMap<
    Node, Node>();
private final ConcurrentSetMap<Node, Node> resourceTypes = new ConcurrentSetMap<Node
    , Node>();
private final ConcurrentSetMap<Node, Triple> customTriplesBySubject = new
    ConcurrentSetMap<Node, Triple>();
private final ReentrantReadWriteLock internalLock = new ReentrantReadWriteLock();
```

Listing 4.7: SBox Locked Triple Insertion (SBox.java)

```
public boolean addTriples(Collection<Triple> triples) {
    readLock(internalLock);
    try {
        return addTriples(triples, false);
    }
    finally {
        readUnlock(internalLock);
    }
}
```

Listing 4.8: SBox Triple Insertion (SBox.java)

```
private boolean addTriples(Collection<Triple> newTriples, boolean flagRDFOnly) {
    boolean changed = false;
    for (Triple triple : newTriples) {
        // reject non-tBox triples
        if (!TBOX_LEVEL.evaluate(triple))
            continue;
        // reject triples that are already known
        if (contains(triple))
            continue;
        // now the triple is considered
        Node subject = triple.getSubject();
        Node object = triple.getObject();
        if (INDIVIDUAL_DEFINITION.evaluate(triple)) {
            addIndividualDefinition(triple);
            storeTriple(flagRDFOnly, triple);
            continue;
        }
        if (TYPE_DEFINITION_TRANSITIVE.evaluate(triple)) {
            types_all.add(subject);
            setDirectSuperType(subject, RDFS_RESOURCE);
            setDirectSubType(RDFS_RESOURCE, subject);
        }
        // link or type definition
        if (!RDFS_RESOURCE.equals(subject) && !subject.equals(object)) {
            setDirectSuperType(subject, object);
            setDirectSubType(object, subject);
        }
        storeTriple(flagRDFOnly, triple);
        changed = true;
        cleaningInterrupt.set(true);
        directTypesCleaned = false;
    }
    // ...
    // }
```

Listing 4.9: SBox Triple Removal (SBox.java)

```
public void remove(Triple triple) {
    writeLock(internalLock);
    try {
        // remove from the triple index
        if (!customTriplesBySubject.remove(triple.getSubject(), triple))
            return;
        // remove individual types
        if (INDIVIDUAL_DEFINITION.evaluate(triple)) {
            removeIndividualTypes(triple.getSubject(), Collections.singleton(triple.
                getObject()));
            return;
        }
        // remove type definition
        removeTypes(Collections.singleton(triple.getSubject()));
    }
    finally {
        writeUnlock(internalLock);
    }
}
```

Triple Distributions and Scent-Lists

The triple distributions contain the number of present triples for each type. The scent-lists contain the amount of pheromones for each type. As both cases are similar, their data type representations are derived from a common base class (DiscreteTypeDistribution), which is basically a concurrent map with type-resources as keys and instances of AtomicDouble as values. The AtomicDouble class is not a part of the java.util.concurrent.atomic package, but was taken from *Java Threads* [53]. It allows concurrent atomic increments, decrements, and multiplications of shared floating point objects[8]. As a result, both, the triple distributions and the scent-lists can be used completely without external synchronization (see listings 4.10, 4.11 and 4.12 for a brief insight).

As the entries in the triple distributions and scent-lists are only used as calculation weights, it not necessary to ensure any specific order of their updates. As long as any increment, decrement, or multiplication is processed at all, the results are considered to be fair enough for the, anyway randomized, algorithms, even if update permutations occur. The gain from this loose synchronization is that all concurrent operations on these distributions are executed remarkably fast.

4.3.4 Conclusion

This gives only a brief overview of the efforts that where taken towards synchronization and minimal mutual exclusion. They not only made the simulation itself more realistic, but they have also made a significant positive impact on the system performance compared to the initial prototype stages, where only the NetLogo modeling language was used. Yet, there were no numeric evaluations made specifically on that issue, since the implementation evolved steadily. These efforts are also the prerequisite for a possible migration from simulation to a real distributed system prototype.

This concludes the descriptions of the simulator and the implementation. For further details, refer to the API documentation, which is available on the appended CD.

[8] In Java it is not possible to use primitive values in maps.

Listing 4.10: AtomicDouble Initialization, from [53] (AtomicDouble.java)

```
private final AtomicReference<Double> atomic;
public AtomicDouble() {
    this(0);
}
public AtomicDouble(double initVal) {
    atomic = new AtomicReference<Double>(Double.valueOf(initVal));
}
```

Listing 4.11: AtomicDouble Multiply And Get, from [53] (AtomicDouble.java)

```
public double multiplyAndGet(double multiple) {
    double origVal, newVal;
    while (true) {
        origVal = atomic.get();
        newVal = origVal * multiple;
        if (compareAndSet(origVal, newVal))
            return newVal;
    }
}
```

Listing 4.12: AtomicDouble Atomic Compare And Set, from [53] (AtomicDouble.java)

```
public boolean compareAndSet(double expect, double update) {
    Double origVal, newVal;
    newVal = Double.valueOf(update);
    while (true) {
        origVal = atomic.get();
        if (origVal.doubleValue() == expect) {
            if (atomic.compareAndSet(origVal, newVal))
                return true;
        }
        else return false;
    }
}
```

Chapter 5

Evaluation

This chapter presents the results of the evaluations conducted on the semantic clustering and retrieval approaches. As these approaches are mainly experimental, self-energizing and also inter-affective, the simulator provides a variety of options to adjust the clustering and retrieval strategies. Thus, the total space of possible configurations is vast and precludes a full covering analysis within a single thesis. Instead, to limit the configuration space, the system evaluation concentrates on major aspects and argues about the transferability of the results. The first object of investigation is the clustering quality. The *spatial sematic entropy* is introduced as a measure of order in the system and the *spatial similarity distribution* indicates the formation of semantic neighborhoods. The second major aspect is triple retrieval. The quality of the sematic trails is evaluated by correlating them with the triple distributions, and the average size of the covered cluster fraction is used to appraise the retrieval performance.

5.1 Clustering Evaluation

5.1.1 Entropy Calculation

Semantic Entropy

In order to quantify the clustering quality, the *spatial sematic entropy* is introduced. This measure generalizes the *spatial entropy* used in SwarmLinda and enhances it to regard semantic similarity. In the approach of Daniel Graff, the spatial entropy considers the distribution of only the four existing distinct templates. He applies the following definition of the entropy H of a discrete random variable X with possible values $x_1, \ldots, x_n$

$$H(X) = E(I(X)) = -\sum_{x \in X} p(x) \log p(x).$$

where $x_1, \ldots, x_n$ are given by the different templates and $p(X = x)$ is defined as the probability of discovering a tuple that matches the template x at the given node.

Using this definition is problematic in terms of general Triple Spaces. Since it requires X to be a *discrete random variable*, the following equation must be fulfilled

$$\sum_{x \in X} p(X = x) = 1 \tag{5.1}$$

In case of a non-restricted template space (i.e. the values of X are all possible templates), this is not the case anymore. When there is an infinite number of templates (n-triples) and any tuple may match various of those templates at the same time, the summed probability of all templates is generally higher that 1. Considering a possible usage of wildcards, $\sum_{n \in \mathbb{N}} p(X = (?_1, \ldots, ?_n))$ is 1 already. Therefore, this approach of entropy calculation is not generally applicable and can only be used for sets of templates with disjunct matches.

In order to quantify the semantic disorder of a general Triple Space, certain modifications are applied. First, the necessity to deal with templates is eliminated. Instead, triples are matched by their primary resource against the finite number of types in the ontology, meaning $\{x_1, \ldots, x_n\} = \mathcal{T}$. Since the triples are indexed by subject, predicate, and object, the spatial sematic entropy is calculated independently for each cluster incarnation.

The second modification is to match resources semantically, regarding the similarities of their types. The probability distribution of X is therefor re-defined in a way, which at the same considers continuous and overlapping matches, and complies with equation 5.1. For this purpose, the spatial sematic entropy integrates the similarity distribution (see section 3.6.1 (Similarity Distribution), page 55) and the triple distribution (see section 3.2 (Task Fulfillment), page 56).

Definition 5.1 (Semantic Entropy) *The* semantic entropy ${}^{n}H_s^{\kappa}$ *of a node* $n \in \mathcal{N}$ *and a cluster index* $\kappa \in \mathcal{K}$ *is the expected value of the information content of the discrete random variable* X *with possible values* $t_1 \ldots t_n \in \mathcal{T}$*. The probability* ${}^{n}p^{\kappa}(X = t)$ *is the similarity-weighted sum of all entries in the local triple distribution* ${}^{n}td^{\kappa}$*, divided by the overall local statement count.*

$${}^{n}H_s^{\kappa} = -\sum_{t \in \mathcal{T}} {}^{n}p_t^{\kappa} \log {}^{n}p_t^{\kappa}$$

$${}^{n}p_t^{\kappa} = {}^{n}p^{\kappa}(X = t) = \sum_{t_i \in \mathcal{T}} \frac{{}^{n}td_{t_i}^{\kappa} \cdot simd_{t_i}^{t}}{|{}^{n}\Omega^{\kappa}|}$$

Lemma 5.2 *Using* ${}^{n}p^{\kappa}$ *as probability distribution, lets* X *become a discrete random variable complying with equation 5.1.*

Proof 5.3 $\forall n \in \mathcal{N}, \kappa \in \mathcal{K}$:

$$\begin{aligned}
\sum_{t \in \mathcal{T}} {}^{n}p^{\kappa}(X = t) &= \sum_{t \in \mathcal{T}} \sum_{t_i \in \mathcal{T}} \frac{{}^{n}td^{\kappa}_{t_i} \cdot simd^{t}_{t_i}}{|{}^{n}\Omega^{\kappa}|} \\
&= \sum_{t_i \in \mathcal{T}} \frac{\sum\limits_{t \in \mathcal{T}} {}^{n}td^{\kappa}_{t_i} \cdot simd^{t}_{t_i}}{|{}^{n}\Omega^{\kappa}|} = \sum_{t_i \in \mathcal{T}} \frac{{}^{n}td^{\kappa}_{t_i} \cdot \sum\limits_{t \in \mathcal{T}} simd^{t}_{t_i}}{|{}^{n}\Omega^{\kappa}|} \\
&\overset{(3.1)}{=} \sum_{t_i \in \mathcal{T}} \frac{{}^{n}td^{\kappa}_{t_i} \cdot 1}{|{}^{n}\Omega^{\kappa}|} = \frac{\sum\limits_{t_i \in \mathcal{T}} {}^{n}td^{\kappa}_{t_i}}{|{}^{n}\Omega^{\kappa}|} \\
&\overset{(3.2)}{=} \frac{|{}^{n}\Omega^{\kappa}|}{|{}^{n}\Omega^{\kappa}|} = 1
\end{aligned}$$

□

Spatial Sematic Entropy

While the semantic entropy is a measure for the resource disorder of a single node, the *spatial sematic entropy* quantifies the resource disorder of the entire network.

Definition 5.4 (Spatial Sematic Entropy) *The* spatial sematic entropy H^{κ}_{sp} *for a cluster index* $\kappa \in \mathcal{K}$ *is the average triple-count-weighted semantic entropy of all nodes* $n \in \mathcal{N}$.

$$H^{\kappa}_{sp} = \sum_{n \in \mathcal{N}} \frac{{}^{n}H^{\kappa}_{s} \cdot |{}^{n}\Omega^{\kappa}|}{|\Theta|}$$

Spatial Sematic Entropy Gain

In SwarmLinda, the spatial entropy is used directly as a quality measure. Yet, this is only appropriate if the clustered data is itself uniformly distributed, in which case the original data entropy is 1. In general ontologies, resource occurrences can be quite imbalanced. While resources tend to appear less frequent as objects than as subjects, the set of predicate resource is generally the smallest. Hence, the subject set, which contains many resources with only few associated triples, has generally a higher immanent entropy than the object set, which contains fewer resources and yet the same number of triples. Regarding to predicates, the immanent entropy is generally the smallest. Therefore, the *spatial sematic entropy gain* is used as the effective cluster quality indicator. The lower the gain, the more relative order is created and the better the clustering quality is.

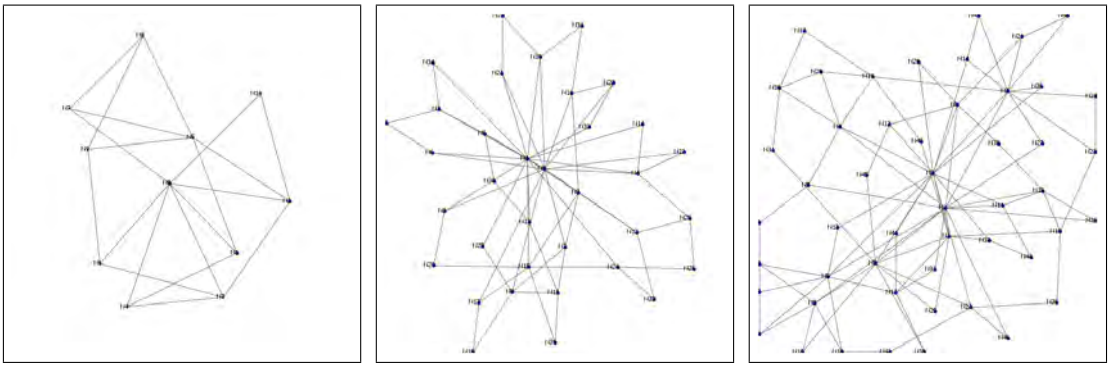
Definition 5.5 (Spatial Sematic Entropy Gain) *The* spatial sematic entropy gain H^{κ}_{g} *for a cluster index* $\kappa \in \mathcal{K}$ *is the quotient of the spatial sematic entropy* H^{κ}_{sp} *and the ontology immanent entropy* H^{κ}_{Θ}.

$$H^{\kappa}_{g} = \frac{H^{\kappa}_{sp}}{H^{\kappa}_{\Theta}}$$

5.1.2 Test Scenarios

Before the results of the entropy gain evaluation for the different similarity measures are discussed, a brief description of the test scenarios is provided. Each scenario is specified by the following parameters:

Network This thesis includes three pre-defined networks, where network 1 consists of 10 nodes, network 2 consists of 34 nodes and network 3 contains 50 nodes (shown in figure 5.1).



(a) Network 1 **(b)** Network 2 **(c)** Network 3

Figure 5.1: Evaluation Networks

Ontology There are also three pre-defined ontologies included in the simulator project. They were derived from the example graphs shown in figure 3.3, page 46, and use *rdfs:Class* as root. Additionally, a number of seven custom properties was added to each ontology and assigned to all classes. Two of these properties are defined to hold resource references, while the others may hold integer values. The latter are established as two plain top-level properties and one top-level property with two sub-properties.

Moreover, a number of ten instances was created for each class. Resource references were added to each individual, with a 25 percent probability (for each of the two properties). Also with a probability of 25 percent, an integer value was assigned for a randomly selected integer property. At last, specific labels and comments were generated for each resource via the default properties *rdfs:label* and *rdfs:comment*.

Similarity Measure Configuration The similarity measure configuration includes the measure itself (*Haase-Siebers-Harmelen*, *Leacock-Chodorow* or *Lin*), the scaling variables α, β, γ, and δ as well as the *degree normalization* option (*on* or *off*)

Cluster Options This includes the *S-Box clustering* option, the *pheromone decay rate*, the *minimum similarity bound*, and the *minimum pheromone level*.

S-Box Clustering	Full Replication	Maximum Ant Count	100
Ant Learning	off	Return Strategy	Shortest Path
Pheromone Decay Rate	0.667	Time-To-Live	15 sec
Minimum Pheromone Level	0.9	ϕ_d	0.9
Minimum Similarity Bound	0.8	ϕ_p	0.9

Table 5.1: Settings for Spatial Sematic Entropy Evaluations

Ant Options The ant configuration includes of the *maximum ant count*, the *time-to-live*, the *ant learning* option (*on* or *off*), the *return strategy* and the randomization variables ϕ_d and ϕ_p.

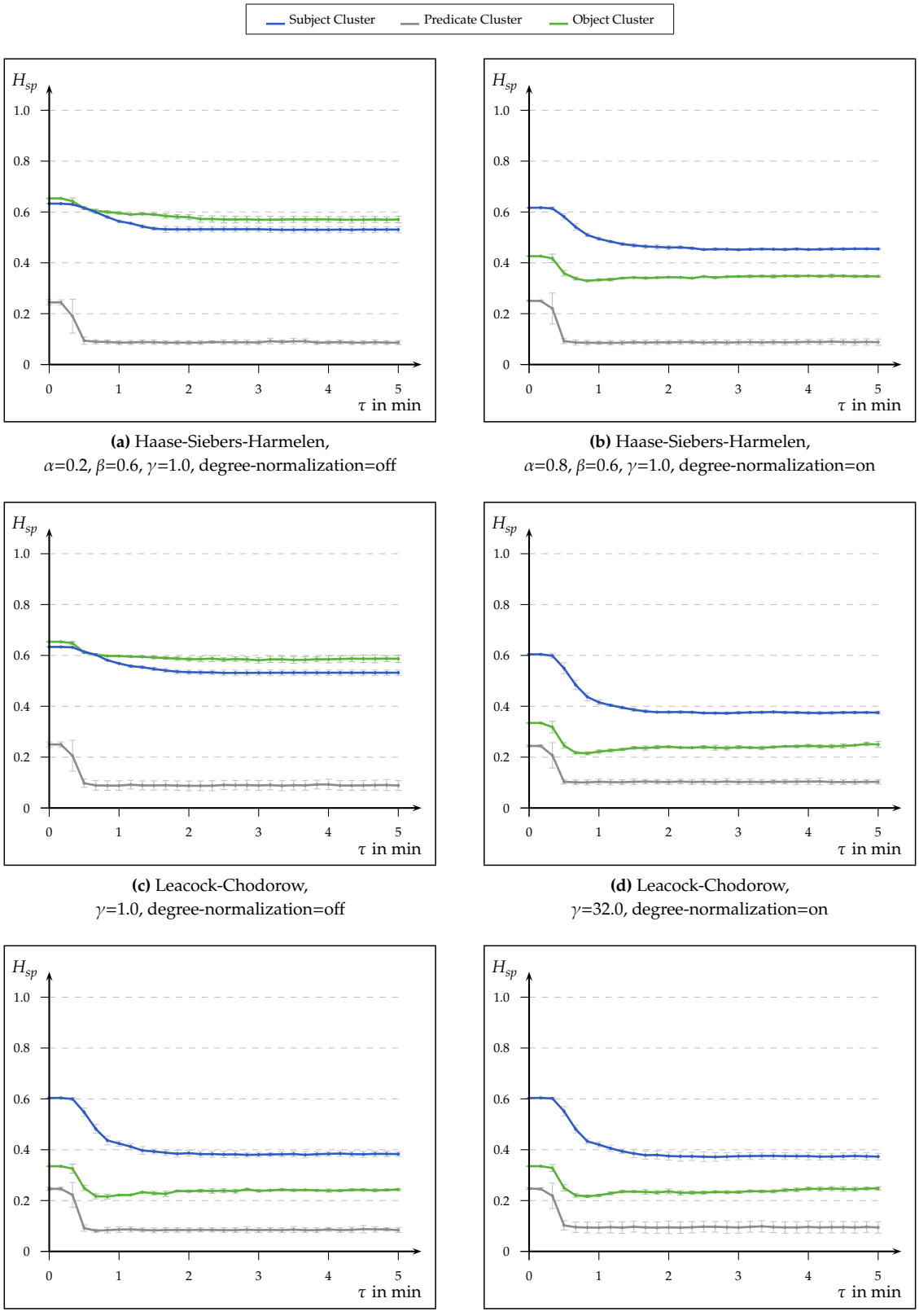
5.1.3 Entropy Evaluation

The spatial sematic entropy gain is evaluated in a series of specific test scenarios. To isolate the entropy gain that actually results from the approached swarm strategies, they are compared to a random walk situation. Therefore, each scenario starts with a random triple insertion phase, in which the ants are configured with ϕ_d=ϕ_p=0. After all triples are located, the default values of ϕ_d and ϕ_p are assigned, the maintenance process is started and the entropy recordings begin.

Besides the variations of the similarity measure configuration, all tests are configured with the settings given in table 5.1. The recorded results as well as their standard deviation after a five-fold repetition are displayed in figure 5.2 and 5.3.

Figure 5.2a and 5.2c show that without scaling and degree normalization, the decrease of the spatial sematic entropy is rather little, when using the distance-based measures Haase-Siebers-Harmelen and Leacock-Chodorow. This is caused by their original flat decrease of similarity (see section 3.4 (Similarity Measure Evaluation), page 45). The more indistinguishable types are, the less additional order can be created via triple redistribution. If the measures are scaled, the entropy decrease becomes more significant, especially in the subject cluster, which is displayed in figure 5.2b and 5.2d. This coherence can be observed more plainly in figure 5.3, which shows the increase of the spatial sematic entropy gain. The measure of Lin, on the contrary, appears to be rather scaling-resistant, since it creates virtually the same results under $\gamma = 1$ and $\gamma = 3$, as shown in figure 5.2e, 5.2f, 5.3e, and 5.3f. This is caused by the fact that Lin bases similarity on the information content of the types. The entropy is itself defined as expected value of the information content. Hence, any scaling of the information content does not change its expected value (and entropy) in relation to the expected value of the information content in the original ontology. The slight differences of the outcomes result only from the applied degree normalization.

These figures represent only a small excerpt of the evaluated configurations, as they only cover the unscaled measures without degree-normalization and the scaled measures with degree-normalization. The results of the other combinations are appended for comparison in figure A.1, page 130, and figure A.2, page 131.



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

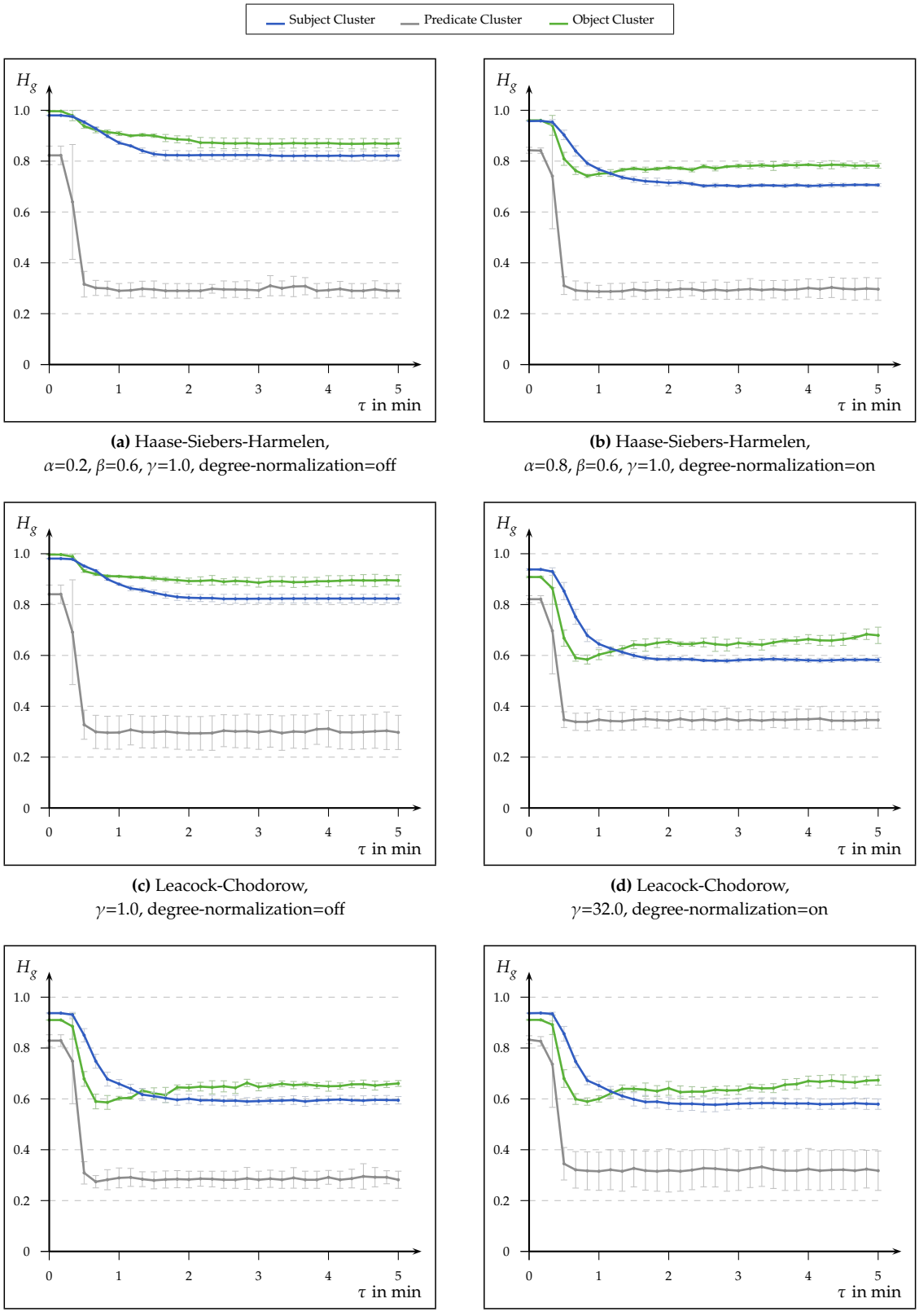
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure 5.2: Measure Spatial Sematic Entropy



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure 5.3: Measure Spatial Sematic Entropy Gain

The entropy evaluations were also conducted under all possible combinations of ontologies and networks. The outcomes are shown in figure A.3, page 132 to figure A.10, page 139. They are comparable to those shown in figure 5.2 and 5.3, yet the final gain of semantic order is generally the higher, the larger the cluster is. This effect results from the fact that a larger number of nodes allows each of them to become thematically more specific.

5.1.4 Local Similarity Gain

The *local similarity gain* is a second indicator for clustering quality. It states the average similarity of co-located triples, relative to the ontology immanent average triple similarity.

Definition 5.6 (Local Similarity Gain) *The* local similarity gain ς^κ *of a cluster and a cluster indicator* $\kappa \in \mathcal{K}$ *is the triple-count-weighted local similarity* ${}^n\varsigma^\kappa$ *of all cluster nodes* $n \in \mathcal{N}$*, relative to the overall triple count and the ontology-immanent average triple similarity* ${}^\Theta\varsigma^\kappa$*.*

$$\varsigma^\kappa = \frac{\sum_{n \in \mathcal{N}} {}^n\varsigma^\kappa \cdot {}^n\Omega^\kappa}{{}^\Theta\varsigma^\kappa \cdot |\Theta|}$$

The local similarity ${}^n\varsigma^\kappa$ of n is defined in section 3.12 (Cluster Maintenance), page 64.

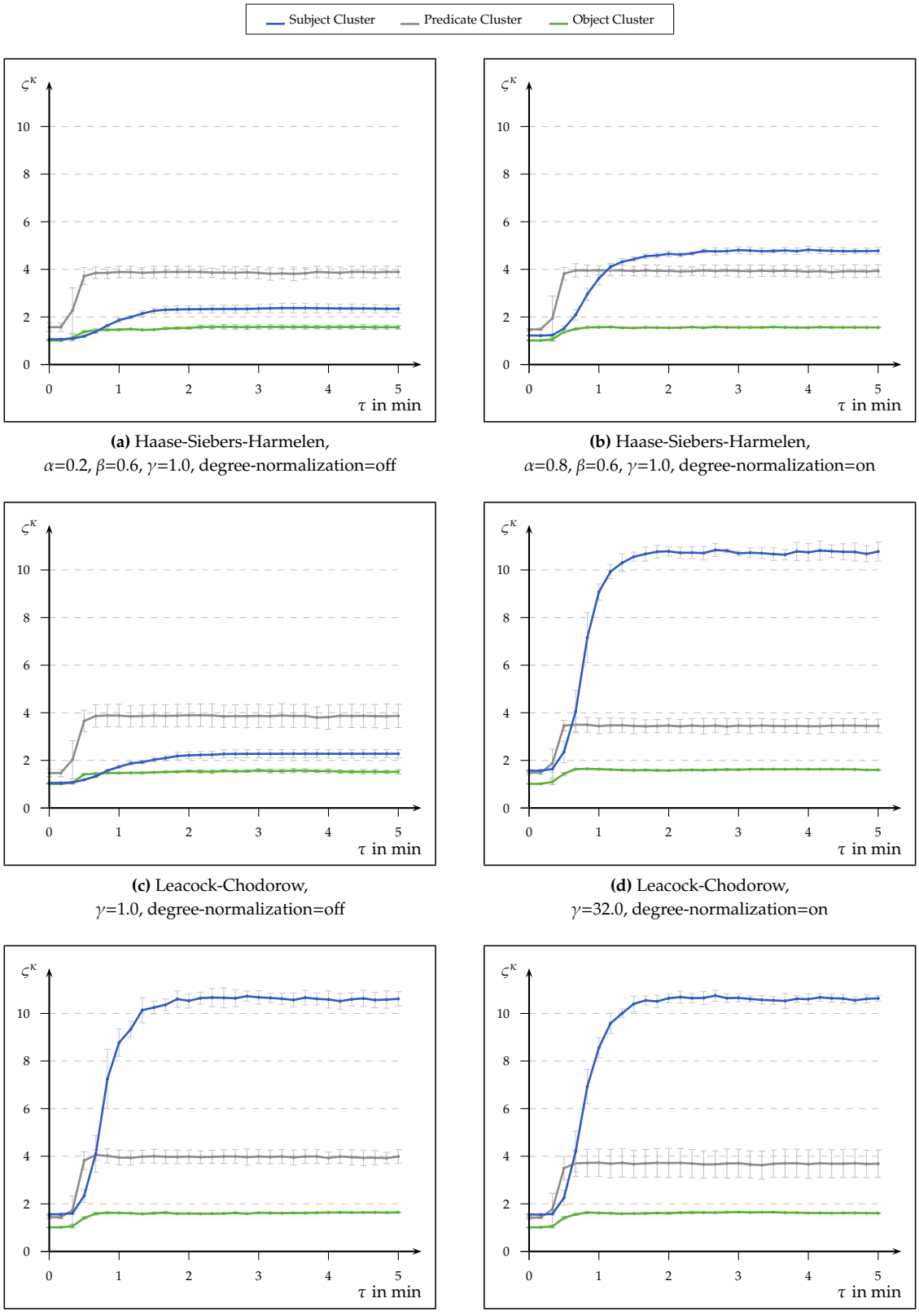
Figure 5.4 shows the progression of local similarity gains in the test scenarios. Again, the unscaled distance-based measures have rather little effect. The scaled Leacock-Chodorow outperforms the scaled Haase-Siebers-Harmelen as it creates a local similarity which is about ten times higher than in the un-clustered ontology. Also in this evaluation, the information-theoretic Lin states virtually the same results, independent of the applied scaling. The results of the additional configuration combinations are appended for comparison in figure A.11, page 140.

Intermediate Conclusions The evaluation results of the spatial sematic entropy gain and the local similarity gain show how the approached strategies increase the semantic order in the cluster and create thematically specialized nodes. They also point out the influence of the selected measure and how measure scaling can be used to increase the clustering quality.

5.1.5 Semantic Neighborhoods

The node local similarity can be generalized to calculate the similarity between arbitrarily nodes. The formation of semantic neighborhoods within the space is analyzed grouping the average pairwise node similarity by network distance.

Definition 5.7 (Node Similarity) *The* node similarity ${}^{n_i,n_j}\varsigma^\kappa$ *of two cluster nodes* $n_i, n_j \in \mathcal{N}$ *and a cluster indicator* $\kappa \in \mathcal{K}$ *is average pairwise similarity of their triples.*



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure 5.4: Measure local similarity gain, Additional Measure Configurations

$$ {}^{n_i,n_j}\varsigma^{\kappa} = \frac{\sum_{t_i \in {}^{n_i}\Omega^{\kappa}} \sum_{t_j \in {}^{n_j}\Omega^{\kappa}} {}^{n_i}\Omega^{\kappa}_{t_i} \cdot {}^{n_j}\Omega^{\kappa}_{t_j} \cdot sim(t_i, t_j)}{|{}^{n_i}\Omega^{\kappa}| \cdot |{}^{n_j}\Omega^{\kappa}|} $$

Hence, the node local similarity ${}^{n}\varsigma^{\kappa}$ is a special case of the general node similarity:

$$ {}^{n}\varsigma^{\kappa} = {}^{n,n}\varsigma^{\kappa} $$

Definition 5.8 (Spatial Similarity) *The* spatial similarity ${}^{\kappa}\rho$ *of a cluster index* $\kappa \in \mathcal{K}$ *is a vector stating the average node similarity, grouped by network distance, where* ${}^{\kappa}\rho_d$ *designates the average node similarity at distance* d*. It is normalized by the average ontology immanent similarity.*

$$ {}^{\kappa}\rho_d = \sum_{n \in \mathcal{N}} \sum_{\substack{n_i \in \mathcal{N} \\ dist(n,n_i)=d}} {}^{n_i}\Omega^{\kappa} \cdot \frac{{}^{n_i,n}\varsigma^{\kappa}}{|\Theta|} $$

Thus, the cluster local similarity gain ς^{κ} becomes a special case of spatial similarity:

$$ \varsigma^{\kappa} = {}^{\kappa}\rho_0 $$

The spatial similarities that occur at the end of the test scenarios are displayed in figure 5.6. Unlike in the entropy evaluation, here the random triple insertion is replaced by the regular insertion, where the values for ϕ_d and ϕ_p are assigned at the beginning and the maintenance process is activated as well. Additionally, an intermediate configuration is evaluated for each measure. For comparison, the spatial similarity after a *random* triple insertion is given in figure 5.5. There, the values at all distances are close to 1. It indicates that the average similarity between nodes is the same at any distance (no semantic neighborhoods). They are also as similar as the triples of un-clustered ontology.

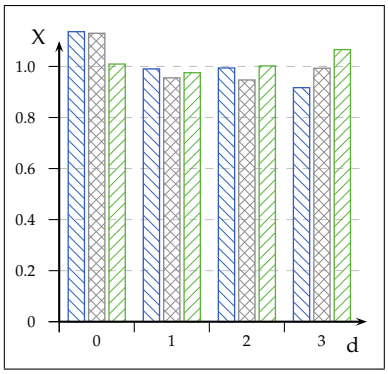
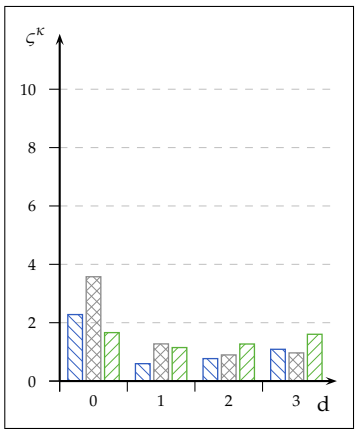
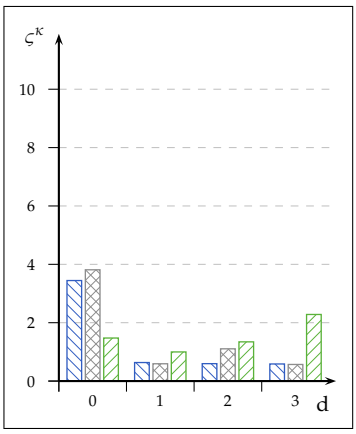


Figure 5.5: Initial Spatial Similarity

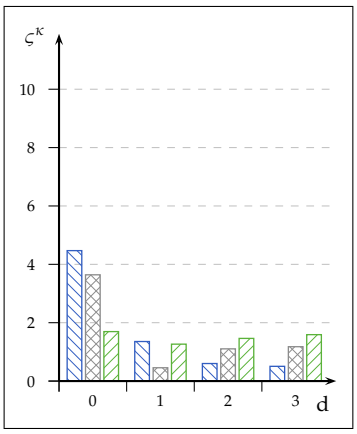
In case the triples are inserted and maintained semantically, the formation of semantic neighborhoods can be observed. Figure 5.6 shows that under all configurations, the highest similarity between nodes occurs at zero distance and then tends to decrease



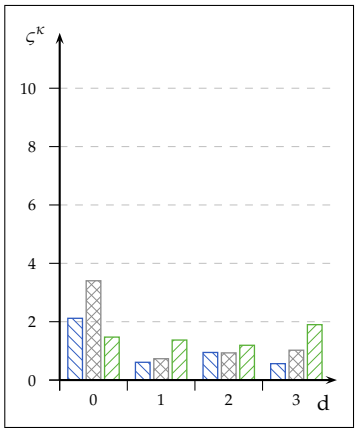
(a) Haase-Siebers-Harmelen, α=0.2, β=0.6, γ=1.0, degree-normalization=off



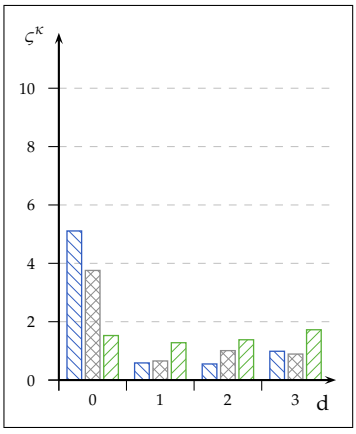
(b) Haase-Siebers-Harmelen, α=0.5, β=0.6, γ=1.0, degree-normalization=on



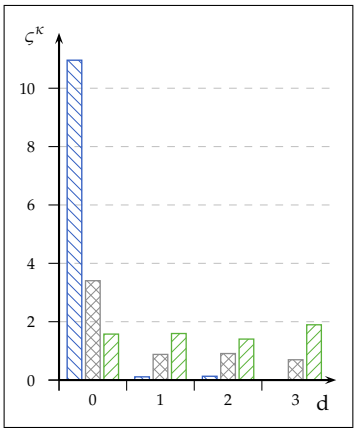
(c) Haase-Siebers-Harmelen, α=0.8, β=0.6, γ=1.0, degree-normalization=on



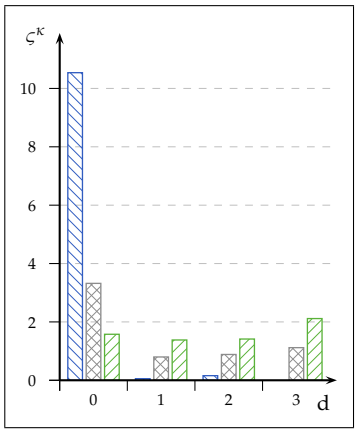
(d) Leacock-Chodorow, γ=1.0, degree-normalization=off



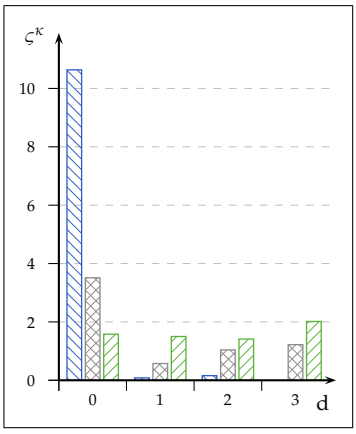
(e) Leacock-Chodorow, γ=4.0, degree-normalization=on



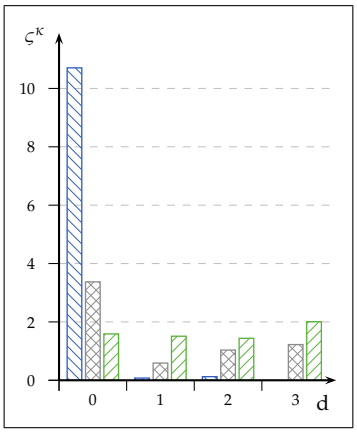
(f) Leacock-Chodorow, γ=32.0, degree-normalization=on



(g) Lin, γ=1.0, degree-normalization=off



(h) Lin, γ=2.0, degree-normalization=on



(i) Lin, γ=3.0, degree-normalization=on

Figure 5.6: Measure Spatial Similarity

with network distance (except from the object cluster, which is discussed in the next section). Yet, the extent of the effect differs, depending on the selected measure and its configuration.

The predicate cluster shows the most stable results. It contains only few resources which are rather dissimilar. As a result, predicate resources can be distributed one property per node, regardless of the similarity measure configuration. On the contrary, the spatial similarity of the object cluster stays quite flat, which indicates rather unspecific and fuzzy neighborhoods. This is caused by the fact, that the relative majority of object resources (1324 of 2875) are strings (labels and comments). Thus, the concentration of strings is very high on each node, as is their local similarity. Since the maintenance process prefers unsuitable resources, the redistribution probability for strings is very low on all nodes. Consequently, the formation of semantic neighborhoods is limited. Yet, this outcome is not totally problematic. It shows that in case many similar resources are added, they do not necessarily concentrate on a single node, which would have to process all request alone (over-clustering), but distribute over the entire network. The most significant formation of semantic neighborhoods can be observed in the subject cluster in all scenarios.

Comparing the configurations of each measure, their influence on the distinctiveness of neighborhoods becomes evident. The results suggest that measure scaling can be used as an immanent facility to adjust the clustering and to prevent anti-over-clustering. This eliminates the need for separate anti-over-clustering strategies like in SwarmLinda.

Intermediate Conclusions The spatial similarity evaluation confirms that the approached strategies are capable of congregating similar concepts and form semantic neighborhoods to a certain degree. It shows how measure scaling can be used to influence the formation of these neighborhoods and furthermore exposes the influence of the original resource distribution on the formation process.

5.1.6 S-Box Clustering

The tests conducted so far had the S-Box clustering deactivated, meaning that all ants make use of a global perspective of the ontology. This is comparable to a fully replicated S-Box-Level with pre-calculated similarities on all nodes. The advantage is clearly, that these calculations can be re-used by all ants, which saves processor time and increases performance. As discussed in section 3.2.2 (T-Box Clustering), page 41, this approach might not be applicable in a real system, since the S-Box-Level could contain many triples and should be distributed as well. This section evaluates the effects of S-Box clustering on performance and the spatial sematic entropy gain.

There are two different S-Box clustering approaches implemented in this thesis. In the first one - *Full Clustering* - at each step all ants merge their own S-Boxes with those of the

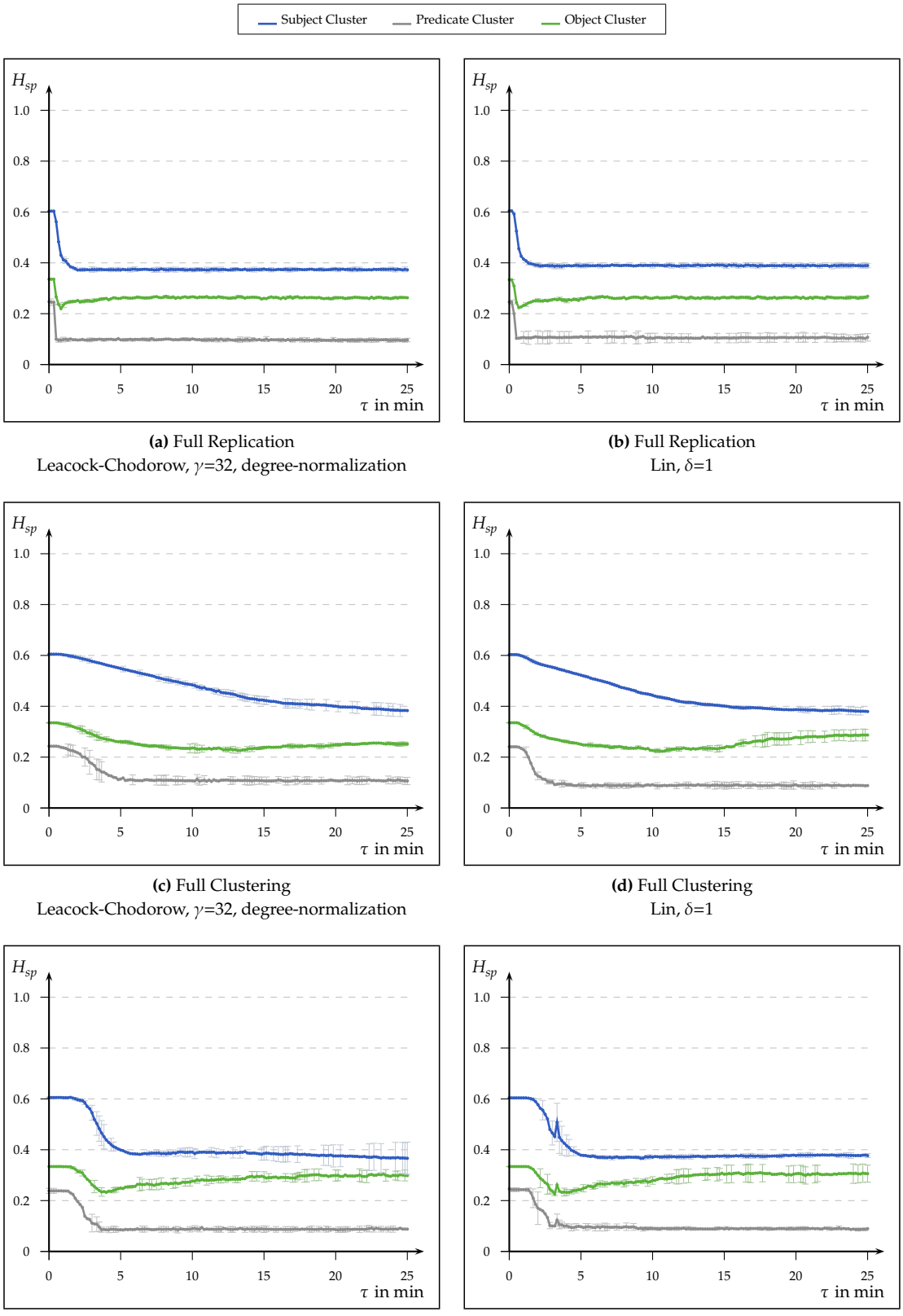
current node and all present ants. Then they re-calculate the similarities and proceed. Anticipating the evaluation results, this strategy induces a lot of calculation time, since similarity calculations require the determination of the ontological distance of all types as well as their LCS and MIS.

The second approach - *Experimental* - is supposed to be a compromise, where the S-Box is still fully distributed, but the costly similarity calculations are not run at every transition by every ant, but instead on the nodes at regular intervals. The results are then re-used by all present ants. The disadvantage of this strategy is that similarities are calculated only for the node-local subset of the S-Box-Level, which could be completely unrelated to the resources of passing ants.

The evaluation of these strategies is conducted by a comparative observation of the spatial sematic entropy gain. The results are shown in figure 5.7. There, the configuration is in all cases the same as in the entropy evaluation, with a random triple insertion phase which is followed by a maintenance phase using one of the clustering strategies. Since the calculation-intensive Full Clustering allows only the usage of a maximum of 50 ants on the test system, this maximum is used in all cases. The Full Clustering is also configured with a higher pheromone decay rate of 0.9 and a higher time-to-live of 150 seconds as an adjustment to the additional calculation time. The tests where performed with Leacock-Chodorow (γ=32) as a distance-based measure and Lin (δ=1) as an information-theoretic measure. Again, they were repeated five times with the experimental standard deviation displayed in the figures.

The results of Leacock-Chodorow and Lin do not significantly differ, event though, in case of Lin, the similarity calculations depend not only on the set of present types, but also on the current triple distribution. The more resources of a specific kind are aggregated, the less specific these type become locally. This results in a lower information content and a lower Lin-similarity towards the other types. Apparently, this effect is not significant enough compared to the impacts of the general S-Box clustering strategy.

Comparing the strategies, the evaluations indicate that the final outcome is similar in all cases. The Full Replication strategy generally creates the fastest entropy decay, while the Full Clustering strategy takes the longest time for semantic redistribution. The performance of the Experimental approach is much closer to the Full Replication, although the S-Box is clustered in the same way as in Full Clustering approach. Its outcome is also identical to the other strategies, which is surprising, since it occurs despite the ants' general lower level of information. This can be explained by the correlation, that a node, the S-Box of which is not related to a passing ant's requirements, also does not host any matching triples. In that case, the similarity of 0 which is stated for the ant's primary resource, repels the ant. Yet, the very same result would have occurred, if the ant had merged its S-Box before, recalculated the similarities and used them to inspect the local triple distribution. In other words, the more relevant the resources of a node are, the more suitable are also its S-Box and pre-calculated similarities.



(a) Full Replication
Leacock-Chodorow, γ=32, degree-normalization

(b) Full Replication
Lin, δ=1

(c) Full Clustering
Leacock-Chodorow, γ=32, degree-normalization

(d) Full Clustering
Lin, δ=1

(e) Experimental
Leacock-Chodorow, γ=32, degree-normalization

(f) Experimental
Lin, δ=1

Figure 5.7: S-Box clustering strategies

Considering these observations, it can be recommended to prefer the Experimental approach to the Full Clustering in a real system. Also, due to its efficiency, the Full Replication might be a good strategy, in case the following assumptions are fulfilled.

- The number of S-Box triples is very small compared to the number of A-Box triples.
- The rate of S-Box operations is rather low compared to A-Box operations.
- The used ontologies are interlinked.

In this respect, it would be worth to conduct statistical evaluations on common Triple Spaces to find out, if these assumptions are fulfilled in general, in which case the Full Replication strategy could be used as default.

Intermediate Conclusions The conducted evaluations confirm that S-Box clustering is possible. The self-organizing aspects of the system ensure that the outcomes are similar to a fully replicated schema, yet they may take a longer time. A fair compromise between scalability and performance is to run similarity calculations for the local S-Box subset on the cluster nodes and to share these results with the passing ants.

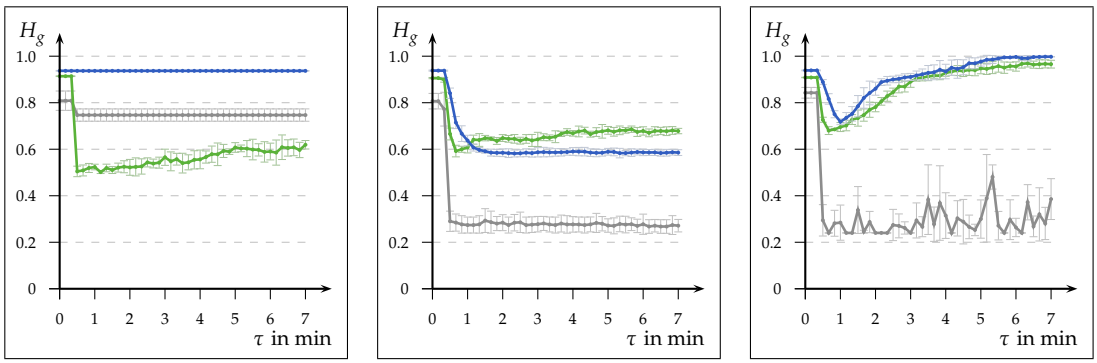
5.1.7 Additional Configuration Options

The additional options evaluated in the next sections are expected to have similar impacts under all measures, ontologies and networks. Therefore, most of them were conducted using Leacock-Chodorow with γ=32, exemplarily. Besides the actual investigated parameter, these tests are same as those used in the entropy evaluation.

Minimum Similarity Bound

Since all node-local resources below the level *minimum similarity bound* × *average local similarity* are considered unsuitable, this option has a significant effect on the maintenance process and the evolving entropy gain. Figure 5.8 displays the results for the minimum similarity bounds of 0.2, 0.8 and 1.1.

In case of a very low minimum similarity bound, the entropy stays rather high (a). This is caused by the set of redistributed triples becoming very small, while the majority of triples stays fixed. On the other hand, if the bound is very high, the set of volatile triples becomes large and the entropy again maintains at a high level (c). In this case, this is not caused by the triples being to stationary, but by the steady redistribution of triples with an already high local similarity. These effects are comparable with the separation process of a substance mixture into distinct products. If the temperature of the mixture is too low (e.g. it is frozen), the separation process stops and the entropy stays high. If the temperature is too high, the particles are constantly re-mixed and the entropy, though



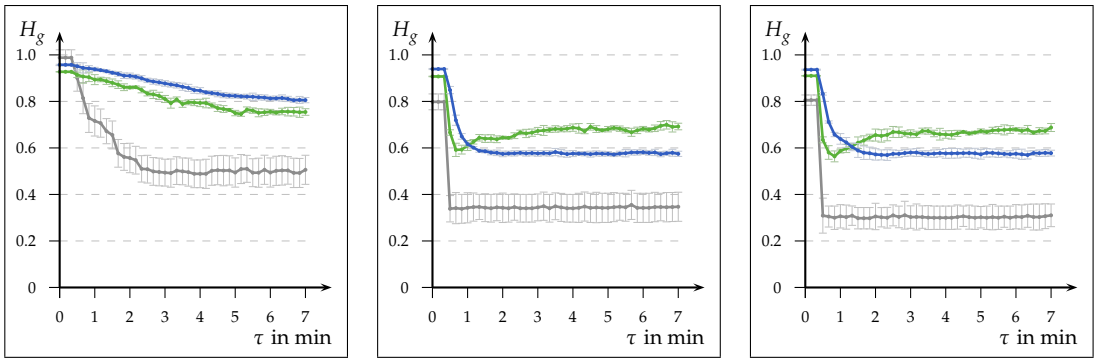
(a) 0.2 **(b)** 0.8 **(c)** 1.1

Figure 5.8: Minimum Similarity Bound Evaluation

larger fluctuations may occur, stays high as well. In the experiments, the optimal *cluster temperature* was found at a minimum similarity bound around 0.8 (c).

Time-To-Live

Quite similar to the minimum similarity is the influence of the ant time-to-live, yet the effect is not as significant. As shown in figure 5.9, if the lifespan is too short to travel long distance, the clustering takes longer (a). If the lifespan is larger than necessary, the ants may search slightly longer, but the clustering result is similar (c). In the specific test setup, a fair configuration appeared around 15 seconds (b).

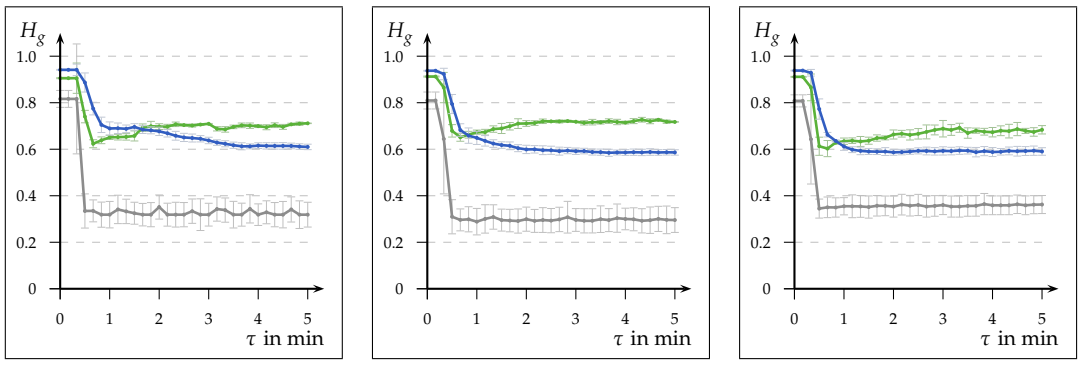


(a) 1.5 sec **(b)** 15 sec **(c)** 1500 sec

Figure 5.9: Time-To-Live Evaluation

Maximum Ant Count

The influence of that option is rather predictable. The more ants are used in the cluster, the faster the triples are processed and the faster the results occur, as it is shown in figure 5.10. The actual limit is of course depending on the technical environment used.

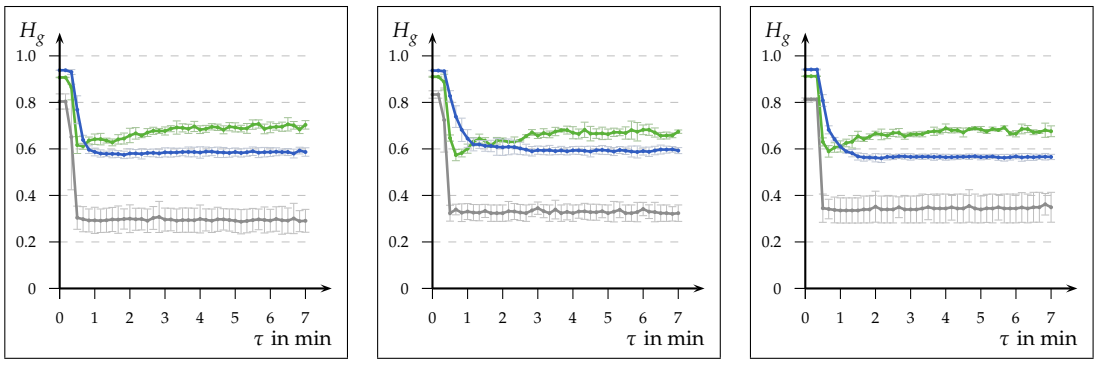


(a) 5 **(b)** 10 **(c)** 100

Figure 5.10: Maximum Ant Count Evaluation

Pheromone Decay Rate

The pheromone decay rate surprisingly has very little effect on the entropy progression (figure 5.11). In fact, this rate merely defines the level, at which pheromones stabilize with the given number of active ants and the configured time-to-live. Since the ants select their paths by a relative comparisons of scents, the actual average pheromone level in the system is not relevant. Yet, the pheromone decay rate defines the cluster response time, since it controls the evaporation of outdated trails.



(a) 0.2 **(b)** 0.666 **(c)** 0.8

Figure 5.11: Pheromone Decay Rate Evaluation

Intermediate Conclusions The evaluation of the additional cluster options shows how their tuning can influence the clustering outcomes. It especially points out the importance of the minimum similarity bound in the maintenance process. Although the influence of the additional options is considered to be similar in other networks and for other ontologies, their *optimal* values depend on the actual hardware, the ontological content, the network size, and the miscellaneous cluster configuration. If, for instance, many ants are used, the pheromones can decay faster and if the ontology homogeneity is high, the average similarity scale should be higher as well. Therefore, further research on the

possibility to adjust these values dynamically and adaptively is suggested. Some possible approaches to achieve this are outlined in section 6.2 (Future Work).

5.2 Retrieval Evaluation

5.2.1 Cluster Processes and Semantic Trail Quality

The basic requirement for a good clustering is that distributed data can be quickly located. In the approaches of this thesis, IN-, RD- and OUT-Ants conduct these searches by following the semantic trails formed by previously spread pheromones. Apparently, it is crucial that these trails do in fact lead to the desired resources to prevent the ants from straying through the space. This issue is evaluated in this section.

In order to quantify the quality of the trail system, the linear correlation between the scents of neighboring nodes and their actual triple distribution is investigated. If this correlation is high, following the scents through the network will most likely lead to the indicated resources. Otherwise, the scent-indicated directions are barely related to the present resources, in which case the ants are might perform random walks as well.

Since both, the scent-lists and the triple distributions, map type-resources to numeric values, the linear relationship between their contents can be calculated using the definition of Bravais and Pearson for the *statistical correlation coefficient* r_{xy} between two random variables X and Y with n given single value pairs $(x_1, y_1), \ldots, (x_n, y_n)$.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \cdot \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Definition 5.9 (Node Triple-Scent Correlation) *The* triple-scent correlation ${}^{n}r^{\kappa}$ *of a node* $n \in \mathcal{N}$ *and a cluster indicator* $\kappa \in \mathcal{K}$ *is the average correlation between the scent-lists of all neighbors of* n *and the triple distribution of* n*. The single correlations are weighted by the total amount of scent.*

$${}^{n}r^{\kappa} = \frac{\sum_{n_i \in {}^{n}\mathcal{N}} |{}^{n,n_i}sc^{\kappa}| \cdot {}^{n_i,n}r^{\kappa}}{\sum_{n_i \in {}^{n}\mathcal{N}} |{}^{n,n_i}sc^{\kappa}|}$$

$${}^{n_i,n}r^{\kappa} = \frac{\sum_{t \in \mathcal{T}} ({}^{n}td_t^{\kappa} - \overline{{}^{n}td_t^{\kappa}})({}^{n_i,n}sc_t^{\kappa} - \overline{{}^{n_i,n}sc_t^{\kappa}})}{\sqrt{\sum_{t \in \mathcal{T}} ({}^{n}td_t^{\kappa} - \overline{{}^{n}td_t^{\kappa}})^2 \cdot \sum_{t \in \mathcal{T}} ({}^{n_i,n}sc_t^{\kappa} - \overline{{}^{n_i,n}sc_t^{\kappa}})^2}}$$

Network	network1	Pheromone Decay Rate	0.667
Ontology	ontology03	Minimum Pheromone Level	0.9
Measure	Leacock-Chodorow	Minimum Similarity Bound	0.8
γ	4	Maximum Ant Count	100
Degree normalization	off	Return Strategy	Shortest Path
S-Box Clustering	off	Time-To-Live	15 sec
Ant Learning	off	ϕ_d	0.9
		ϕ_p	0.9

Table 5.2: Settings for Cluster Triple-Scent Correlation Evaluations

Definition 5.10 (Cluster Triple-Scent Correlation) *The* cluster triple-scent correlation r^κ *for a cluster index* $\kappa \in \mathcal{K}$ *is the average triple-scent correlation* ${}^n r^\kappa$ *of all nodes* $n \in \mathcal{N}$.

$$r^\kappa = \frac{1}{|\mathcal{N}|} \sum_{n \in \mathcal{N}} {}^n r^\kappa$$

To investigate the influence of the different cluster activities independently, the semantic trail evaluation is divided into 5 phases. Phase 2-5 run 10 minutes each with a continuous recording of the cluster triple-scent correlation every 30 seconds.

Phase 1 inserts the triples without any additional processes running. At the end of this phase, the cluster triple-scent correlation is recorded as initial value.

Phase 2 starts the maintenance process.

Phase 3 stops the maintenance process and starts the template generation with the return strategy Taken Path.

Phase 4 runs the template generation with the return strategy Shortest Path.

Phase 5 investigates the combination of maintenance and template generation at a process balance rate of 0.67.

As the tendentious influence of the maintenance and template generation process is supposed to be independent of the actual measures, network or ontology, one exemplary test scenario is used for evaluation (see table 5.2 for configuration). The average results after 5 iterations and their standard deviation are displayed in figure 5.12.

Since OUT-Ants place their scents directly at the drop node, its neighbors, and the path they used, the correlation between the scents and triple distributions is very high at the end of the insertion phase. After that, the maintenance process assigns unsuitable resources to OUT-Ants for relocation. As a result, the correlation of scents and triples decreases, since these ants spread pheromones only for transient triples, while the original trails are unmaintained and evaporate. In the worst case, the maintenance process creates a low entropy, but leaves no qualified information about the locations of stable triples. Figure 5.12 shows how the correlation actually decreases close to zero in this phase.

To compensate this immanent decrease of scent quality, the cluster can generate random templates in times of no external activities to keep the sematic trails trained

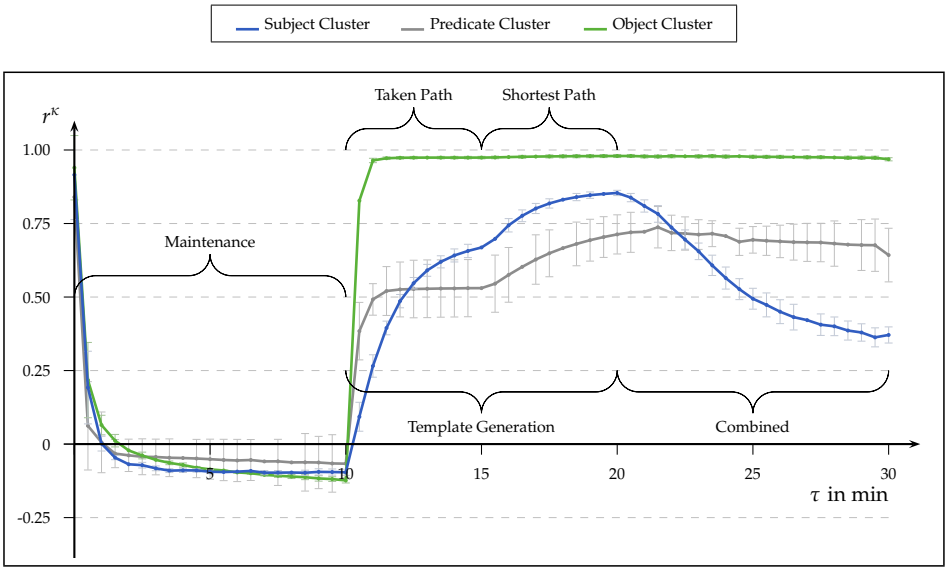


Figure 5.12: Cluster Triple-Scent Correlation

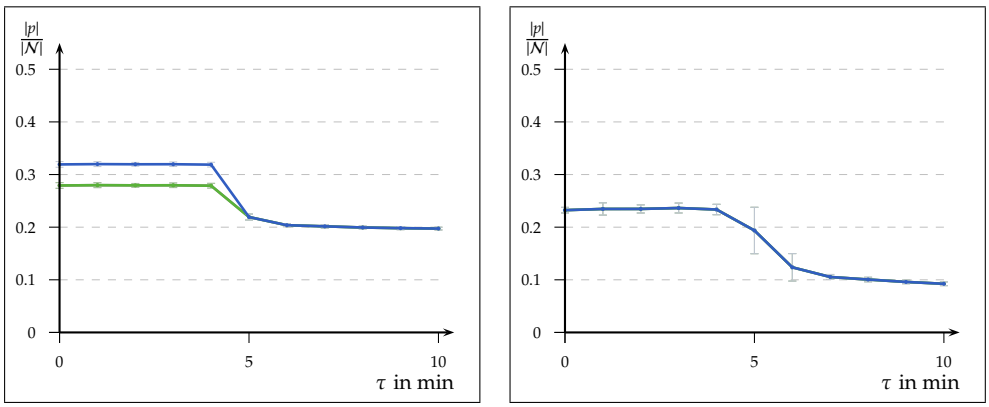
(template generation process). This process is activated in the third phase. As displayed in figure 5.12, the template generation is capable of restoring the triple-scent correlation to a fair degree. The correlation raises even more, when the experimental return strategy Shortest Path is used in phase 4, since the ants only mark the optimal path between the request node and the retrieved match.

Finally, phase 5 investigates the mixture of the two processes. The process balance of 0.67 (2 template ants for each relocating ant) appeared to be a fair compromise on the test system as the triple-scent correlation stays high and there is an appropriate capacity left for triple relocation to create a low spatial sematic entropy.

5.2.2 Search Path Evaluation

The final evaluation conducted investigates the average search path length of template ants. It utilizes the template generation, as it already creates continuous random requests. The recording of the results, presented in figure 5.13, started after a 5 min maintenance time. To evaluate the improvement of retrieval performance caused by the semantic trails, ϕ_p was set to 0 during the first 10 minutes to simulate a random walk and then reset to 0.9, so that the ants primarily use the scents for triple tracking. Since a request is fulfilled, when the first ant successfully returns, only the path lengths of these first ants were monitored, ignoring any subsequent results. In the figure, the average search path length is given relative to total number of nodes.

As the results in figure 5.13a show, when using a random walk, the ants have to cover approximately one third of the network to find their match. This is already an improvement to the regular expected coverage of 0.5 for random searches and is due to



(a) Network 1 **(b)** Network 3

Figure 5.13: Average Template Search Path Length

the retrieval parallelization. Shown in the second phase, the usage of the sematic trails improves the retrieval performance even more, as the average network coverage drops to approximately one fifth.

When using a larger network for the same ontology, the average network coverage is reduced even more, as displayed in figure 5.13b. This is caused by the fact that in a larger network the nodes become more specific, and so do the trails. Of course, this does not imply that the retrieval quality can be scaled arbitrarily by using larger networks, as the coverage of one sixth of a larger network, for instance, may still take longer than traversing one fifth of a small network.

Intermediate Conclusions The scent and retrieval evaluations prove that following the implemented self-energizing semantic trails significantly increases retrieval performance. Moreover, this performance generally decreases less worse with network size than random walks. The evaluations furthermore suggest to sustaining the adaptiveness of the cluster by establishing a continuous training facility, which compensates for triple relocation and the evaporation of scents.

Chapter 6

Conclusions and Future Work

This chapter provides the final conclusions of this thesis. It recapitulates its accomplishments, summarizes the made observations, and addresses existing limitations. Furthermore, this chapter includes an outline of possible improvements and suggestions for future work.

6.1 Conclusions

6.1.1 Accomplishments

In this thesis, a variety of approaches for semantic triple clustering and retrieval in distributed RDF-Spaces was introduced, implemented, and evaluated. Using the NetLogo simulator framework, a de-centralized, scalable and adaptive cluster network was created, which is maintained by a swarm of virtual ants.

It was demonstrated how these ants are capable of dynamically clustering triples by the semantic similarity of their resources. Therefore, they were provided with basic skills of transitive type inference and similarity determination, using common distance- and information-theoretic similarity measures. In the space, these ants operate completely autonomous and independent, making strictly local decisions only.

Additionally, a form of loosely coupled communication between these agents was established by adapting the natural pheromone-based stimulus of ants, to create a system of sematic trails. Similar to real ant colonies, these trails improve the efficiency of resource locating and are themselves amplified depending on the success rate of the routed ants.

Regarding the internal structures of ontologies, the necessity to distinguished between the different levels of abstraction in RDF was discussed and two separate approaches for clustering the A-Box- and T-Box-level were provided. In the latter, the T-Box was limited to the similarity-relevant type hierarchy definitions only (S-Box). By a comparative evaluation, three different strategies for S-Box clustering were analyzed, which range from

a complete replication to a complete distribution with similarity calculations conducted by the ants or at the nodes.

In order to quantify the clustering and retrieval quality, appropriate indicators were introduced, such as the spatial sematic entropy gain and the spatial similarity distribution, which investigate the sematic distribution of triples, as well as the cluster triple-scent correlation and the average search path length, which allow a measure-independent retrieval evaluation. Since the approaches of this thesis are mostly experimental, different configuration options are provided and their influences on quality and performance were evaluated.

6.1.2 Observations

The conducted evaluations investigated how the quality of clustering and retrieval is influenced by the selected measure, the chosen clustering strategy, and the system configuration. Moreover, these tests also indicated the impacts of external parameters, such as the used ontology or the network size. Especially the distributions of subject-, predicate- and object-resources in the un-clustered ontology determine the primary clusterability of the triples. If these resources are very heterogenous, the amount of achievable additional order is higher than in case of resource homogeneity. The second major external influence is the network size. A large network allows more specific nodes and likewise creates more specific trails. Yet, the possible search space becomes larger as well, which, at a certain point, can significantly decrease retrieval performance. So far, there is no strategy established to determine the optimal network size depending on the triple count and entropy. As it is also not certain, if these two parameters are even sufficient enough, further research in this area is suggested for future works.

The most important cluster configuration is the similarity measure. There, only the information-theoretic measure of Lin creates an appropriate clustering quality without further modifications. On the contrary, the original similarity stated by the distance-based measures of Haase-Siebers-Harmelen and Leacock-Chodorow is too high, which results in a rather homogenous perception of resources and a low clusterability. The disadvantage of Lin is, nonetheless, that its similarity depends on both, the S-Box and the A-Box content, while Haase-Siebers-Harmelen and Leacock-Chodorow depend on the S-Box only. As the A-Box content may change rapidly, Lin consequently requires more frequent re-calculations.

To compensate for the high resource similarity of the distance-based measures, degree normalization and measure scaling were used to create results comparable to Lin. They additionally allow to configure the distinctiveness of the semantic areas. Considering all evaluation results, the usage of Leacock-Chodorow can be generally recommended, as it is based on the S-Box only, creates a fair semantic triple distribution, and is easy to configure. For specialized ontologies with a low average degree,

Haase-Siebers-Harmelen is recommended, since the influence of concept height and distance be adjusted independently using its scaling variables α and β.

The S-Box clustering strategy is another crucial clustering factor. Besides the Full Clustering strategy, in which the ants conduct all similarity calculations themselves, the Experimental strategy, which shifts this issue to the nodes, was implemented and compared. As discussed in section 5.1.6 (S-Box Clustering), the Full Clustering induces a lot of calculations, many of which are redundant. The Experimental approach instead establishes an S-Box clustering, which is both scalable and efficient.

The most important factor for retrieval quality is the appropriateness of the semantic trail system. While triples and resources are constantly redistributed by the maintenance process to ensure adaptiveness and to decrease the spatial sematic entropy, it is necessary to keep the trails up-to-date and to sustain the retrieval responsiveness. In this respect, the conducted evaluations verify the importance of a continuous cluster training facility, to prevent the trails from vanishing and misleading. The trail formation itself is configurable by the return strategy. There, the strategy of simply returning the taken path (without loops) proved to already create a high correlation between trails and resources, due to its self-energizing character. Yet, the results can still be improved if the optimal (shortest) return path is used. This strategy is not yet implemented in a distributed way, yet a suggestion of a possible realization is contained in section 6.2.3.

The tendentious influence of the additional options such as time-to-live, maximum ant count, and process balance rate have been illustrated in the evaluation as well. Since they are basically system-dependent, no general recommendation can be given for these values. Instead, in a real system, they must be adjusted properly, depending on the actual hardware capacity. The minimum similarity bound has the most significant impact on the redistribution process. Regarding the clustering of a relative majority of similar resources, it is suggested to replace this bound with a dynamic adjustment in future works, which is discussed in section 6.2.1.

6.1.3 Limitations

Naturally, the presented approaches are not free of limitations and insufficiencies. The most obvious of them is the lack of triple updates, which are not provided because they are not part of the originally proposed LINDA primitives. Fortunately, updates can be implemented easily as a modification of IN-Ants that, instead of removing a locked triple, simply modify its values.

For the same reasons, the distributed cluster also does not support a sophisticated query language like SPARQL. However, implementing such is not as straightforward because quantifiers such as *All* must cover the whole space and cannot be distributed. Since other quantifiers like *Any* or *Exists* could be evaluated without a full network traversal,

additional research on the possibility of distributed and concurrent query evaluations for Triple Spaces is recommended.

At last, the capacity of the simulator is limited itself. Using network of 50 nodes, means to simulate the processor and memory power of 50 machines, which is at about the limit of the test system. In order to evaluate extremely large networks and ontologies, the migration to a real distributed system and a capable hardware environment is suggested. The first step in that direction has already been undertaken, when the decision was made to implement true agent concurrency.

6.2 Future Work

6.2.1 Maintenance Improvements

In this work, the set of redistributable resources is defined by all resources with a local similarity that is below a certain level (minimum similarity bound) of the average local similarity. This is already a dynamic approach, as this level raises the more similar resources are aggregated. In case there is a relative majority of similar resources in the ontology, this may result in a situation, where there is a high concentration of these resources on every node in the cluster. Due to their high concentration, the chance of relocation is little for these resources. Thus, they in fact do annex the nodes, which prevents other kinds of resources to congregate at a spot of their own.

To solve this problem, it must be ensured that there is a certain possibility of relocation for every resource, even if it appears to be properly located already (local optimum). Unfortunately, while experimenting with such strategies, the cluster entropy raised in all cases. As described earlier, this is comparable with a temperature that is too high to separate a mixture into distinct products, as all particles are constantly mixed again and again.

Therefore, it is suggested to evolve an adaption of the *simulated annealing* strategy. In such an approach, the redistribution could start with a high similarity bound, which is then constantly lowered until the cluster freezes. Anytime new triples are added, they would heat up the cluster again until it once more cools down and all triples are resettled. This approach would have two major advantages. The first one is that in a hot cluster, the probability of resource relocation would be higher than zero for all present triples. This allows to leave a local optimum so that, for instance, a high concentration of similar resources at two different nodes could evolve into an even higher concentration of the same resources on a single node. The second advantage is that by cooling down, the cluster entropy would not remain at a high level, but decrease while the upper similarity bound drops and fewer and fewer mobile resources exist. At the end, the cluster maintenance could even stop completely, saving processor time and data traffic, until again triples are added or removed.

6.2.2 Correlation-Based Decisions

In the proposed strategies, probabilistic decisions, like the drop decision or the path selection, are based on the concentration of a given resource within a distribution of resources (triple distribution or scent-list). This concentration is calculated by applying similarity-weights to the given entries and summarizing them. As it is required that such a concentration (and the derived probability) is between 0 and 1, the similarity weights are normalized to sum up to 1 as well. This causes the similarity of a resources towards its own direct type to be less than 1, since there must always be some similarity left to rank the other types. Hence, in case a resource is compared to a distribution with only resources of exactly the same type, the stated concentration is lower than 1, as there exist no entries for other types and the applied similarity weights are lost. This effect becomes less significant when the similarity between types decreases fast, but it still exists and prevents resources to be placed at a completely suitable node with a probability of 1 (probability randomization not considered).

This problem can possibly be solved with a complete different approach for concentration calculation. Instead of using discrete entries for distributions - like adding an amount of pheromones for the direct parent types of a resource - the continuous similarity distribution of that type could be added instead. To match a resource against such a distribution, the correlation would be calculated and projected from $[-1, \ldots, 1]$ to $[0, \ldots, 1]$. As a result, when matching a resource against a completely suitable distribution, a correlation of 1 would occur and the probability of dropping the triple or of choosing that node would be 1 as well.

This approach was discussed in the early stages of this thesis, yet it was rejected for the time being because of two reasons. First, the projection of the correlation is not straightforward. Since a negative correlation would suggest to reject the node, it should not be projected to a probability of $[0, \ldots, 0.5]$. Secondly, the correlation could be less useful in the case of mixed resources, since it is expected to rapidly decrease to zero. So, in order to create a base for comparison, the similarity weight approach was implemented. Yet, in future works, the correlation-based approach might be investigated as well.

6.2.3 Distributed Shortest Path

As discussed earlier, using the return strategy Shortest Path creates a higher cluster triple-scent correlation and a better clustering quality, compared to the Taken Path strategy. The Shortest Path strategy was implemented by using NetLogo internals to determine the shortest path between two nodes from a global perspective. A distributed version could implemented in a possible bachelor or master thesis by adapting the ACO approach. In an additional cluster layer, a new kind of ants could maintain a system of *node scents* to indicate and maintain the shortest route between any two cluster locations. The sematic

ants could then follow these scents to return to their requestor and update the semantic trails along their way.

6.2.4 Further Suggestions

As the final suggestions for future works, certain enhancements of the triple clustering and retrieval are proposed. While so far, support for typed templates is provided, the system could easily be extended to process multiple triple requests as well. Furthermore, templates could be extended with similarity ranges so that resources within a certain semantic distance could be retrieved. In such a scenario, precision and recall could be applied as common quality indicators.

Secondly, improvements in the field of similarity determination are also possible. In the current implementation, similarity is defined by the ontological distance or the information content. In future works, it could be extended by a facility that considers arbitrary RDF properties which express inter-type relatedness. Such an extended conception would allow to cluster resources not only by similarity, but also under certain aspects, for instance, to congregate *books* of the same *author*, *lectures* who teach at the same *institute*, or resource *elements* of the same *container*.

At last, and concluding this thesis, research on the issue of *micro-reasoning* in a distributed ontology is suggested, to find out if small additional reasoning capabilities of ants or nodes can improve semantic clustering and retrieval by inferring additional statements and resource relationships.

List of Figures

1.1 Spatial triple distribution in SwarmLinda URI-Extensions. 13
1.2 Spatial triple distribution in Semantic SwarmLinda. 14

2.1 RDF graph, derived from S-P-O triples 16
2.2 RDF container membership represented by S-P-O triples 18
2.3 RDF List definition represented by S-P-O triples 19
2.4 A simple one-to-one communication pattern 20
2.5 The four major stages in the development of semantic web services 22
2.6 Abstraction levels in a typical RDF-S ontology 24
2.7 The similarity measure of Wu and Palmer 27
2.8 Allowed changes of directions in Hirst and St-Onge 28
2.9 A fragment of the WordNet taxonomy with probabilities and information content assigned 29
2.10 Pseudocode for a swarm-based solution of the TSP 33
2.11 Control elements and visualization area of the SwarmLinda simulator . . 34
2.12 An illustration of a general JavaSpaces™ application 36

3.1 The triple Space architecture 43
3.2 Using URI-clustering for type identification 45
3.3 The example graphs used for similarity measure evaluation 46
3.4 Average similarity results for example graphs 47
3.5 Similarity measure decrease in example graphs 48
3.6 Average similarity in example graphs 49
3.7 Wu-Palmer-approximation using Haase-Siebers-Harmelen 50
3.8 Similarity distribution in example graphs 52

3.9 Similarity distribution in example graphs with applied degree-normalization 53

4.1 Dependencies of the simulator projects 68
4.2 The Network Generator interface 70
4.3 The simulator main view 72

5.1 Evaluation Networks 88
5.2 Measure Spatial Sematic Entropy 90
5.3 Measure Spatial Sematic Entropy Gain 91
5.4 Measure local similarity gain, Additional Measure Configurations 93
5.5 Initial Spatial Similarity 94
5.6 Measure Spatial Similarity 95
5.7 S-Box clustering strategies 98
5.8 Minimum Similarity Bound Evaluation 100
5.9 Time-To-Live Evaluation 100
5.10 Maximum Ant Count Evaluation 101
5.11 Pheromone Decay Rate Evaluation 101
5.12 Cluster Triple-Scent Correlation 104
5.13 Average Template Search Path Length 105

A.1 Measure Spatial Sematic Entropies, Additional Measure Configurations . 130
A.2 Measure Spatial Sematic Entropy Gain, Additional Measure Configurations 131
A.3 Measure Spatial Sematic Entropy Gain, Network 1, Ontology 1 132
A.4 Measure Spatial Sematic Entropy Gain, Network 2, Ontology 1 133
A.5 Measure Spatial Sematic Entropy Gain, Network 3, Ontology 1 134
A.6 Measure Spatial Sematic Entropy Gain, Network 1, Ontology 2 135
A.7 Measure Spatial Sematic Entropy Gain, Network 2, Ontology 2 136
A.8 Measure Spatial Sematic Entropy Gain, Network 3, Ontology 2 137
A.9 Measure Spatial Sematic Entropy Gain, Network 2, Ontology 3 138
A.10 Measure Spatial Sematic Entropy Gain, Network 3, Ontology 3 139
A.11 Measure Local similarity gain 140

List of Listings

4.1 ABox Fields 80
4.2 ABox Initialization 80
4.3 ABox Triple Locking 80
4.4 ABox Conditional Triple Removal 80
4.5 ABox Unconditional Triple Removal 81
4.6 SBox Fields 81
4.7 SBox Locked Triple Insertion 81
4.8 SBox Triple Insertion 82
4.9 SBox Triple Removal 82
4.10 AtomicDouble Initialization, from [53] 84
4.11 AtomicDouble Multiply And Get, from [53] 84
4.12 AtomicDouble Atomic Compare And Set, from [53] 84

List of Definitions

3.1 Similarity Distribution 55
3.2 Triple Distribution 56
3.3 Type Concentration 56
3.4 Drop Probability 57
3.5 Effective Drop Probbalility 57
3.6 Scent-List 57
3.7 Semantic Suitablity 58
3.8 Transition Probability 58
3.9 Effective Transition Probability 58
3.10 Most Dissimilar Resources 64
3.11 Local Similarity 64
3.12 Node Local Similarity 64
5.1 Semantic Entropy 86
5.4 Spatial Sematic Entropy 87
5.5 Spatial Sematic Entropy Gain 87
5.6 Local Similarity Gain 92
5.7 Node Similarity 92
5.8 Spatial Similarity 94
5.9 Node Triple-Scent Correlation 102
5.10 Cluster Triple-Scent Correlation 102

List of Acronyms

ACM	*Association for Computing Machinery, Inc.* – Association for Computing Machinery, Inc. [29]	2
ACO	*Ant Colony Optimization* – A class of algorithm that solve problems by simulated ants and pheromones to find short paths in graphs.	2, 3, 6
API	*Abstract Programming Interface* – Generally an extendable set of interfaces and implementations of a programming language intended for a common specific usage.	2, 4
HTTP	*Hypertext Transfer Protocol* – A protocol developed by the W3C and the IETF for exchanging Hypertext documents via a series of request - response interactions between a client an a server.	2
IDE	*Integrated Development Environment* – Application designed for Software Development integrating various tools	4
JAR	*Java archive* – An archive containing Java classes and possibly meta information and resources	2, 4
JDK	*Java Development Kit* – includes the JRE and additional tools for developing Java applications	4
JINI	*Java Intelligent Network Infrastructure* – A Java-based framework for the development of distributed applications.	2
JRE	*Java Runtime Environment* – The minimal required Java installation to run programs in the Java Virtual Machine	4
LCS	*least common subsumer* – Generally the node with the maximal depth that subsumes two given nodes. In some cases also the considered maximum value is not taken from a node's depth but on some other characteristic value such as weighted distance from the given nodes or information content (*MIS*).	2–5

MIS	*most informative subsumer* – The common subsumer of two or more nodes in a graph that has the maximum information content.	2–5
OWL	*Web Ontology Language* – Is a W3C recommendation for a family of languages to express domain knowledge in a machine processable way. Its syntax is based of RDF and RDF-S, which it extends for more expressiveness [15].	1, 2, 4
POM	*Project Object Model* – A Maven description of Java project, defining, among others, its source folders, resources, and dependencies	4
RDF	*Resource Description Framework* – A family of W3C recommendations for a metadata facility [8]	1–4, 6
RDF-S	*RDF-Schema* – A vocabulary extension of the RDF that provides a mechanism of defining ontologies by describing types as well as their properties and hierarchy [14]	1, 2, 4
RDQL	*RDF Data Query Language* – A query language for extracting information from RDF graphs [54]	2
REST	*Representational State Transfer* – Refers to a system architecture where state between processes is shared via persistent resources addressable by global unique identifiers.	2
SOAP	*Simple Object Access Protocol* – An XML-based protocol used for an interoperable exchange structured and typed message content [55].	2
SPARQL	*SPARQL Protocol and RDF Query Language* – An RDF query language (recursive acronym)	1, 2, 4, 6
TSP	*Traveling Salesperson Problem* – The problem of finding the smallest path in a graph that includes all nodes.	1, 2
URI	*Uniform Resource Identifier* – A character sequence to identify web resources. An URI consists of the following parts: *<URI-scheme>:<scheme specific part>*. Frequently used URI-schemes are *http*, *ftp*, *file* or *mailto* the scheme specific parts of which are also hierarchical [56].	1–3

UUID	*Universally Unique Identifier* – A 128-bit ($\approx 10^{38}$) random number being unique across space and time by considering, among others, the generator's IEEE 802 MAC address and generation time. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network [57].	3
W3C	*World Wide Web Consortium* – An international organization founded for the development of World Wide Web standards (recommendations) [58]	1, 2
WWW	*World Wide Web* – A network of interlinked HTML documents and other resources that is accessible by the Internet. It was founded by Tim Berners-Lee in 1989, Geneva, Switzerland [58].	1, 2
XML	*Extensible Markup Language* – A specification for markup language creation. Specific XML languages describe an interoperable format for structured data in form of nested tags with properties, property values and content [60].	2, 4

Index

A-Box, 23, 40–43, 55, 62, 75, 99, 106, 107
Ant Learning, 73, 89, 103
Augustin, Anne, 12, 14, 15, 39, 40, 42, 43, 61, 63
Average Search Path Length, 104, 107

Boyer-Moore, 22

Cluster Triple-Scent Correlation, 103, 104, 107, 110, 115

Degree Normalization, 50, 51, 55, 73, 88, 89, 103, 107
Drop Probability
    Randomized, 57, 115
Drop Probability, 34, 39, 56–58, 115

Effective Transition Probability, 58, 115
eLinda, 38
EVAL-Primitive, 21, 36

GigaSpaces XAP, 37, 63
Graff, Daniel, 4, 12, 14, 15, 33–35, 39, 40, 67, 70, 85

Haase-Siebers-Harmelen, 25, 45, 46, 49–51, 73, 88, 89, 92, 107, 108

IBM, 37
IN-Ant, 35, 42, 43, 54, 61–64, 71, 75, 77, 102, 108
IN-Primitive, 12, 20, 34, 35, 61, 62
Information Content, 28, 31
INP-Primitive, 20

Java™ programming language, 12, 33, 36–38, 66–69, 78, 83, 116, 117
JavaSpaces™, 36, 37

Leacock-Chodorow, 46, 50, 51, 88, 89, 92, 97, 99, 103, 107
Levenshtein, 22, 69
Lin, 31, 50, 51, 88, 89, 92, 97, 107
LINDA, 4, 12, 13, 15, 19–21, 35–39, 60, 63, 108
Local Similarity Gain, 92–94, 115, 140

Maintenance Process, 43, 64, 73–75, 79, 96, 103, 104, 108
Markov Chain, 76
Maximum Ant Count, 43, 64, 74, 89, 100, 101, 103, 108
Measure Scaling, 107
Minimum Pheromone Level, 73, 88, 89, 103
Minimum Similarity Bound, 64, 73, 88, 89, 99–101, 103, 108, 109

NetLogo, 12, 33, 34, 60, 66–71, 78, 83, 106, 110
Node Similarity, 92, 94, 115
    Node Local Similarity, 64, 73, 77, 92, 94, 115
    Node Local Similarity Gain, 76
    Similarity Similarity, 76, 94–96, 115
Node Triple-Scent Correlation, 102, 103, 115

OpenSpaces, 37
OUT-Ant, 34, 35, 41, 43, 54–56, 60–62, 64, 71, 74–77, 102, 103
OUT-Primitive, 12, 20, 34
OWL
    OWL DL, 19

OWL Full, 19
OWL Lite, 18

Pheromone Decay Rate, 65, 73, 88, 89, 101, 103
Process Balance Rate, 74, 103, 108

Rada-Mili-Bicknell-Blettner, 46, 49
RD-Ant, 35, 43, 44, 54, 61–65, 75, 77, 102
RD-Primitive, 12, 20, 21, 60
RDF
Data-Level, 23
RDF-S-Level, 23, 41
rdf:Alt, 17
rdf:Bag, 17
rdf:first, 18
rdf:List, 18
rdf:nil, 18
rdf:Property, 16–18, 44, 54
rdf:rest, 18
rdf:Seq, 17
rdf:type, 12, 17, 23, 44, 54
rdf:XMLLiteral, 16
rdfs:Class, 16, 18, 44, 54, 88
rdfs:comment, 17, 88
rdfs:Container, 17
rdfs:Datatype, 16
rdfs:domain, 17
rdfs:label, 17, 88
rdfs:Literal, 16
rdfs:range, 17
rdfs:Resource, 16, 23
rdfs:subClassOf, 17, 23, 44, 54
rdfs:subPropertyOf, 17, 23, 44, 54
Schema-Level, 23, 41, 42
RDP-Primitive, 21
Return Strategies
Shortest Path, 60, 73, 103, 104, 110
Taken Path, 59, 73, 103, 110
Return Strategy, 73, 89, 103, 104, 108, 110

S-Box, 42–44, 60, 62, 65, 73, 74, 79, 96, 97, 99, 106, 107
Clustering, 60, 73, 88, 89, 96, 98, 99, 103, 106, 108
S-Box Clustertering
Experimental Strategy, 73, 97, 99, 108
Full Clustering Strategy, 73, 96, 97, 99, 108
Full Replication Strategy, 73, 89, 97, 99
Scent-List, 39, 42–44, 57–60, 65, 73, 77, 83, 102, 110, 115
Semantic Cluster, 43, 49, 61, 66, 68, 70, 71, 73, 74
Semantic Entropy, 85–87, 115
Spatial Sematic Entropy, 43, 55, 76, 85–87, 89, 90, 101, 103, 104, 107–109, 115, 130
Spatial Sematic Entropy Gain, 76, 87–89, 91, 92, 96, 97, 99, 107, 115, 131–139
Semantic Suitablity, 40, 41, 57, 58, 115
Semantic Web Service, 4, 13, 21, 22, 38
Semantic Web, 11, 38
Similarity Distribution, 51, 55, 56, 61, 64, 86, 115
Spatial Similarity Distribution, 85, 107
Suffix Array, 23
Suffix Tree, 23
Sun Microsystems, 36
SwarmLinda, 4, 13, 33, 34, 39, 69, 70, 85, 87, 96
Semantic Extensions, 14
URI-Extensions, 13

T-Box, 23, 41, 42, 106
TBox
Clustering, 40, 41, 73, 96
Template Generation Process, 43, 74, 75, 103, 104
Time-To-Live, 34, 35, 55–57, 59, 61, 63, 71, 74, 77, 89, 97, 100, 101, 103, 108
Transition Probability, 58, 115
Tripcom Project, 38
Triple Distribution, 44, 56, 64, 77, 83, 85, 86, 97, 102, 103, 110, 115

Spatial Triple Distribution, 13, 14
Triple Space, 13–15, 32, 39–41, 43, 45, 54, 55, 58, 61, 63, 66, 78, 86, 99, 109
Triple Space Computing, 13, 21
TSpaces, 37, 63
Tuple Space, 4, 12, 19–21, 33–39
Type Concentration, 56, 115

URI-Ant, 44
URI-Cluster, 13, 42, 44, 45, 61, 68, 71, 74

Web Service, 4, 21, 38
Wu-Palmer, 46, 50

XML-Schema, 15, 16
XMLSpaces.NET, 37, 38
XPath, 38
XQuery, 38

Z-Algorithm, 22

Bibliography

Bibliography

[1] Reto Krummenacher et al. "WWW or What is Wrong is with Web Services". in: *Proc. 3rd European Conf. on Web Services*. edited by Welf Löwe and Jean-Philippe Martin-Flatin. IEEE Computer Society, 2005, pages 235–243. URL: http://members.deri.at/~retok/publications/41_Krummenacher_R.pdf. see pages 4, 19, 21, 22.

[2] Daniel Graff. "Implementation and Evaluation of a SwarmLinda System". Masterarbeit. FU Berlin, 2008. see pages 4, 12, 33, 34, 39, 70.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web – A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities". in: *Scientific American* 284.5 (May 2001), pages 34–43. URL: http://www.sciam.com/article.cfm?id=the-semantic-web. see page 11.

[4] SemanticWeb.org. *SemanticWeb.org Main Page*. URL: http://semanticweb.org/wiki/Main_Page. see page 11.

[5] W3C Consortium. *W3C Semantic Web Activity*. URL: http://www.w3.org/2001/sw/. see page 11.

[6] David Gelernter. "Generative communication in Linda". in: *ACM Trans. Program. Lang. Syst.* 7.1 (1985), pages 80–112. ISSN: 0164-0925. DOI: http://doi.acm.org/10.1145/2363.2433. see pages 12, 19, 20.

[7] Anne Augustin. "RDF Extensions of SwarmLinda". Bachelorarbeit. FU Berlin, 2008. see pages 12, 39, 40, 42.

[8] Ivan Herman, Dan Brickley, and Ralph Swick. *Resource Description Framework (RDF)*. W3C Recommendation. February 2004. URL: http://www.w3.org/RDF/. see pages 15, 117.

[9] Graham Klyne, Jeremy Carroll, and Brian McBride. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. February 2004. URL: http://www.w3.org/TR/rdf-concepts. see page 15.

[10] Frank Manola, Eric Miller, and Brian McBride. *RDF Primer*. W3C Recommendation. February 2004. URL: http://www.w3.org/TR/rdf-primer/. see pages 16–19.

[11] David C. Fallside and Priscilla Walmsley. *XML Schema Part 0: Primer Second Edition*. W3C Recommendation. October 2004. URL: http://www.w3.org/TR/xmlschema-0/. see page 15.

[12] Henry S. Thompson et al. *XML Schema Part 1: Structures Second Edition*. W3C Recommendation. October 2004. URL: http://www.w3.org/TR/xmlschema-1/. see page 15.

[13] Paul V. Biron and Ashok Malhotra. *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. October 2004. URL: http://www.w3.org/TR/xmlschema-2/. see page 15.

[14] Dan Brickley, Ramanathan V. Guha, and Brian McBride. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. February 2004. URL: http://www.w3.org/TR/rdf-schema/. see pages 16–18, 117.

[15] Deborah L. McGuinness and Frank van Harmelen. *OWL Web Ontology Language Overview*. W3C Recommendation. February 2004. URL: http://www.w3.org/TR/owl-features/. see pages 18, 117.

[16] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. *OWL Web Ontology Language Guide*. W3C Recommendation. February 2004. URL: http://www.w3.org/TR/owl-guide/. see page 18.

[17] Sean Bechhofer et al. *OWL Web Ontology Language Reference*. W3C Recommendation. February 2004. URL: http://www.w3.org/TR/owl-ref/. see page 18.

[18] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C Recommendation. February 2004. URL: http://www.w3.org/TR/owl-semantics/. see page 18.

[19] Dieter Fensel. "Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information". in: *Proc. of the IFIP Int'l Conf. on Intelligence in Communication Systems*. edited by Finn Arve Aagesen, Chutiporn Anutariya, and Vilas Wuwongse. volume 3283. Lecture Notes in Computer Science. Springer Verlag, 2004, pages 43–53. URL: http://dip.semanticweb.org/documents/Triple-basedComputing-SWS2004.pdf. see pages 19, 21, 22.

[20] George Wells. *Coordination languages: Back to the future with Linda*. 2005. URL: http://wcat05.unex.es/Documents/Wells.pdf. see pages 20, 21.

[21] George Wells, A. G. Chalmers, and P. G. Clayton. "Linda implementations in Java for concurrent systems: Research Articles". in: *Concurr. Comput. : Pract. Exper.* 16.10 (2004), pages 1005–1022. ISSN: 1532-0626. DOI: http://dx.doi.org/10.1002/cpe.v16:10. URL: http://www.cs.bris.ac.uk/Publications/Papers/2000380.pdf. see pages 21, 36–38.

[22] Elena Simperl, Reto Krummenacher, and Lyndon Nixon. "A Coordination Model for Triplespace Computing". in: (June 2007). URL: http://www.tripcom.org/docs/coordination07_paper.pdf. see pages 21, 38.

[23] A. Magkanaraki et al. *Benchmarking RDF Schemas for the Semantic Web*. 2002. see page 24.

[24] Valentina Cordi et al. *An Ontology-Based Similarity between Sets of Concepts*. see page 24.

[25] R. Rada et al. "Development and application of a metric on semantic nets". in: *IEEE Transaction on Systems, Man and Cybernetics* 19 (January 1989), pages 17–30. see pages 24, 25.

[26] S. Castano et al. "Semantic Information Interoperability in Open Networked Systems". in: *Proc. of the Int. Conference on Semantics of a Networked World (ICSNW), in cooperation with ACM SIGMOD 2004*. Paris, France 2004, pages 215–230. see page 25.

[27] P. Haase, R. Siebers, and F. von Harmelen. "Selection in Peer-to-Peer Networks with Semantic Topologies". in: *In Proc. of International Conference on Semantics of*. 2004. see pages 25, 26.

[28] Association for Computing Machinery. *The ACM Computing Classification System [1998 Version]*. 1998. URL: http://oldwww.acm.org/class/1998/. see page 25.

[29] Association for Computing Machinery. *Association for Computing Machinery*. URL: http://www.acm.org/. see pages 25, 116.

[30] Claudia Leacock and Martin Chodorow. "Combining Local Context and WordNet Similarity for Word Sense Identification". edited by C. Fellbaum. in: *An Electronic Lexical Database* (1998), pages 265–283. see page 26.

[31] Zhibiao Wu and Martha Palmer. "Verb semantics and lexical selection". in: 1994, pages 133–138. see pages 26, 27.

[32] Philip Resnik. "Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language". in: *Journal of Artificial Intelligence Research* 11 (1999), pages 95–130. see pages 26, 28, 29, 31.

[33] Graeme Hirst and David St-Onge. "Lexical chains as representations of context for the detection and correction of malapropisms". in: Toronto, Canada: MIT Press, 1995. see pages 27, 28.

[34] Dekang Lin. "An information-theoretic definition of similarity". in: *Proc. 15th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1998, pages 296–304. see pages 28, 29, 31.

[35] Michael L. McHale. *A Comparison of WordNet and Roget's Taxonomy for Measuring Semantic Similarity*. URL: http://citeseer.ist.psu.edu/580719.html. see page 28.

[36] Jay J. Jiang and David W. Conrath. "Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy". in: *In Proceedings on International Conference on Research in Computational Linguistics*. 1997. see page 28.

[37] A. Budanitsky. *Semantic Distance in WordNet: An Experimental, Application-oriented Evaluation of Five Measures*. 2001. see page 28.

[38] Enda Ridge and Daniel Kudenko. *A Study of Concurrency in the Ant Colony System Algorithm*. URL: http://citeseer.ist.psu.edu/760698.html. see page 32.

[39] V. Cicirello and S. Smith. *Insect societies and manufacturing*. 2001. URL: http://citeseer.ist.psu.edu/cicirello01insect.html. see pages 32, 33.

[40] Northwestern University. *NetLogo Homepage*. URL: http://ccl.northwestern.edu/netlogo/. see page 33.

[41] Sun Microsystems. *Java Spaces™ Specification*. URL: http://java.sun.com/products/jini/2.0.2/doc/specs/html/jsTOC.html. see page 36.

[42] Qusay H. Mamoud. *Getting Started With JavaSpaces Technology: Beyond Conventional Distributed Programming Paradigms*. June 2005. URL: http://java.sun.com/developer/technicalArticles/tools/JavaSpaces/. see page 36.

[43] GigaSpaces Technologies Inc.. *Giga Spaces Homepage*. URL: http://www.gigaspaces.com/. see page 37.

[44] IBM Research. *TSpaces Intelligent Connectionware*. URL: http://www.almaden.ibm.com/cs/TSpaces/index.html. see page 37.

[45] Robert Tolksdorf, Franziska Liebsch, and Duc Minh Nguyen. *XMLSpaces.NET: An Extensible Tuplespace as XML Middleware*. technical report B-03-08. 2003. URL: ftp://ftp.inf.fu-berlin.de/pub/reports/tr-b-03-08.pdf. see page 37.

[46] Massimo Marchiori and Liam Quin. *W3C XML Query (XQuery)*. W3C Recommendation. 2007. URL: http://www.w3.org/XML/Query/. see page 38.

[47] Anders Berglund et al. *XML Path Language (XPath) 2.0*. W3C Recommendation. January 2007. URL: http://www.w3.org/TR/xpath20/. see page 38.

[48] Tripcom Project. *Tripcom Project Homepage*. URL: http://www.tripcom.org/. see page 38.

[49] Tripcom Project. *Tripcom Project Presentation*. April 2008. URL: http://www.tripcom.org/docs/TripcomPPT.pdf. see page 38.

[50] Sun Microsystems. *The Source for Java Developers*. 2008. URL: http://java.sun.com/. see page 66.

[51] The Eclipse Foundation. *Eplise Project Site*. 2008. URL: http://www.eclipse.org/. see page 66.

[52] Michael Mitzenmacher and Eli Upfal. *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. ISBN: 0521835402. see page 76.

[53] Scott Oaks and Henry Wong. *Java Threads*. September 2004. URL: http://ccl.northwestern.edu/netlogo/. see pages 83, 84, 114.

[54] Andy Seaborne. *RDQL - A Query Language for RDF*. W3C Member Submission. January 2004. URL: http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/. see page 117.

[55] W3C Consortium. *Latest SOAP versions*. W3C Recommendation. April 2007. URL: http://www.w3.org/TR/soap/. see page 117.

[56] URI Planning Interest Group. *URIs, URLs, and URNs: Clarifications and Recommendations 1.0*. W3C Recommendation. September 2001. URL: http://www.w3.org/TR/uri-clarification/. see page 117.

[57] P. Leach, M. Mealling, and R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. July 2006. URL: http://tools.ietf.org/html/rfc4122. see page 118.

[58] W3C Consortium. *World Wide Web Consortium*. URL: http://www.w3.org. see page 118.

[59] DERI Innsbruck at the Leopold-Franzens-Universität Innsbruck, Austria, DERI Galway at the National University of Ireland, Galway, Ireland, BT, The Open University, and SAP AG. *Web Service Modeling Ontology (WSMO) Submission*. W3C Submission. April 2005. URL: http://www.w3.org/Submission/2005/06/.

[60] Liam Quin. *Extensible Markup Language (XML)*. W3C Recommendation. URL: http://www.w3.org/XML/. see page 118.

[61] Frank Leymann. *Space-based Computing and Semantics: A Web Service Purist's Point-Of-View*. technical report 5. University of Stuttgart, Institut für Architektur von Anwendungssystemen, 2006. URL: http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2006-05&mod=0&engl=0&inst=IAAS.

[62] C. Bussler et al. "D21. v0. 1 WSMX Triple-Space Computing". in: *WSMO Working Draft June* (June 2005). URL: http://www.wsmo.org/TR/d21/v0.1.

[63] Dieter Fensel et al. *Queues Are Spaces - Yet Still Both Are Not The Same?* Technical report. 2007. URL: http://www.spacebasedcomputing.org/fileadmin/files/SBC-QueuesAreSpaces_20070511.pdf.

[64] Ray Richardson, Alan F. Smeaton, and J. Murphy. *Using WordNet as a Knowledge Base for measuring semantic similarity between words*. technical report CA-1294. Dublin, Ireland 1994. URL: http://citeseer.ist.psu.edu/richardson94using.html.

[65] Internet Engineering Task Force. *The Internet Engineering Task Force*. URL: http://www.ietf.org/.

[66] W3C Consortium. *HTML 4.01 Specification*. W3C Recommendation. December 1999. URL: http://www.w3.org/TR/html401/.

[67] W3C Consortium. *Web Services Description Language (WSDL) 1.1*. W3C Recommendation. March 2001. URL: http://www.w3.org/TR/wsdl.

[68] Patrick Hayes and Brian McBride. *RDF Semantics*. W3C Recommendation. February 2004. URL: http://www.w3.org/TR/rdf-mt/.

[69] Dave Beckett and Brian McBride. *RDF/XML Syntax Specification (Revised)*. W3C Recommendation. February 2004. URL: http://www.w3.org/TR/rdf-syntax-grammar/.

[70] Steve Bratt. *Semantic Web, and Other Technologies to Watch*. W3C Presentation. 2007. URL: http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(1).

[71] M. Steyvers and J. Tenenbaum. *Small Worlds in Semantic Networks*.

[72] P Bouquet et al. "Asking and answering semantic queries". in: *In Proc. of Meaning Coordination and Negotiation Workshop (MCNW-04) in conjunction with International Semantic Web Conference (ISWC-04), 2004*. 2004.

[73] S. Patwardhan, S. Banerjee, and T. Pedersen. "Using Measures of Semantic Relatedness for Word Sense Disambiguation". in: *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*. Mexico City, Mexico 2003, pages 241–257. URL: http://citeseer.ist.psu.edu/patwardhan03using.htm.

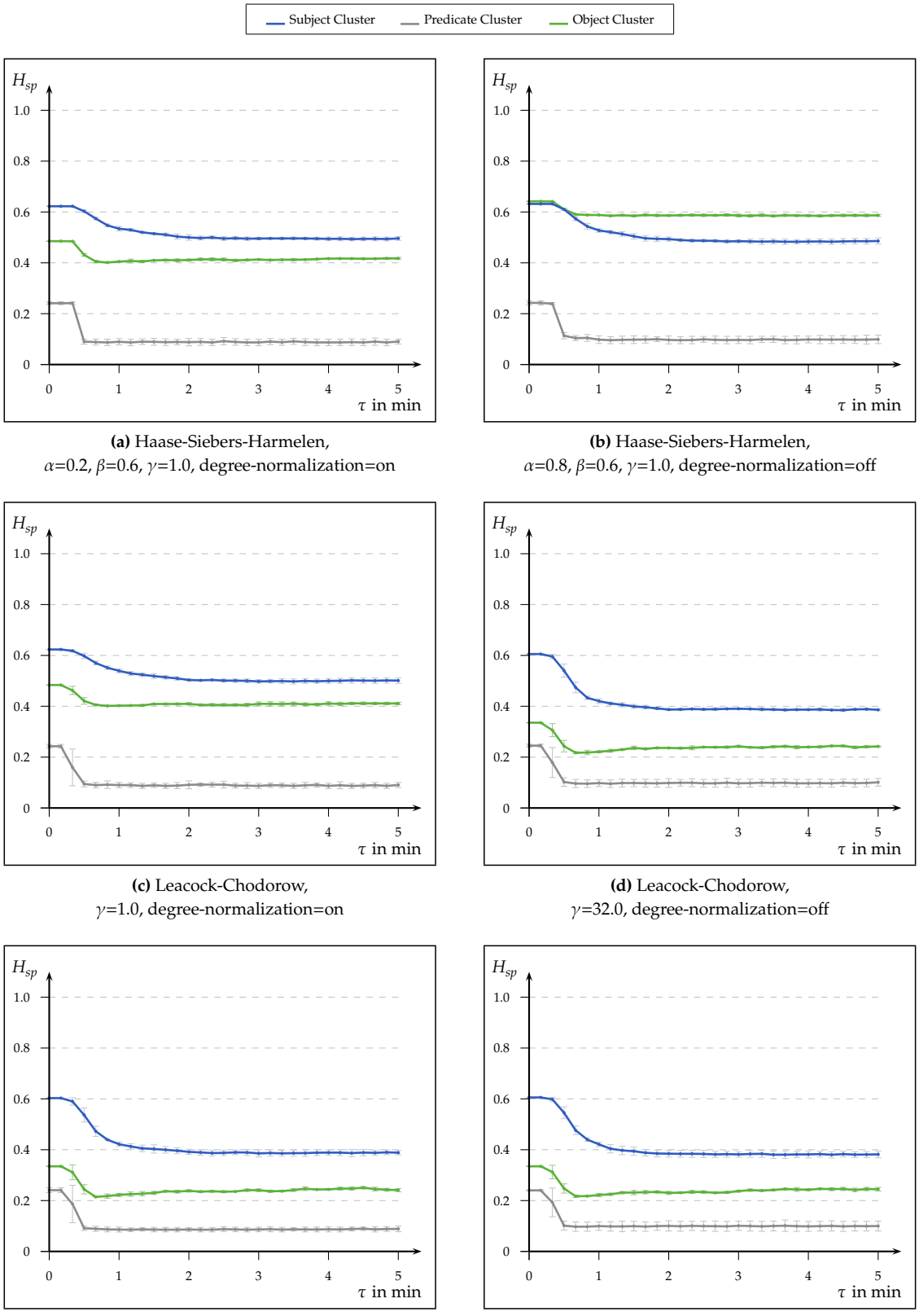
[74] Rafael Grote. "Entwurf und Implementierung eines ontologiebasierten Ähnlichkeitsalgorithmus für semistrukturierte Dokumente". Bachelorarbeit. FU Berlin, 2006. URL: http://page.mi.fu-berlin.de/grote/swpatho/bsc.pdf.

[75] Jeffrey M. Bradshaw. "An Introduction to Software Agents". in: *Software Agents*. edited by Jeffrey M. Bradshaw. AAAI Press / The MIT Press, 1997, pages 3–46. URL: http://citeseer.ist.psu.edu/bradshaw97introduction.html.

[76] Matteo Casadei Luca. *Collective Sorting Tuple Spaces*. URL: http://citeseer.ist.psu.edu/765064.html.

Appendix A

Further Evaluation Results



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=on

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=off

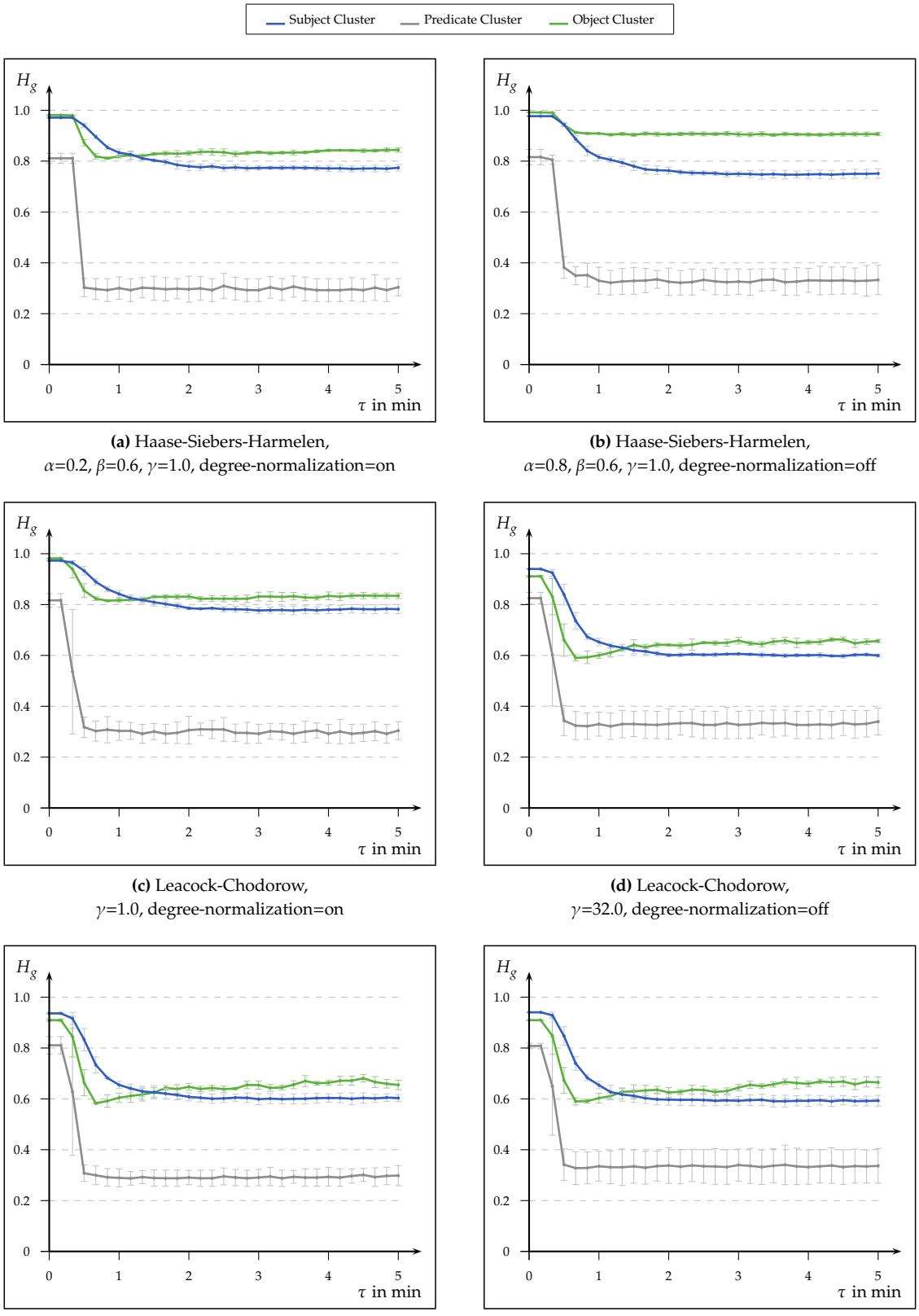
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=on

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=off

(e) Lin,
γ=1.0, degree-normalization=on

(f) Lin,
γ=3.0, degree-normalization=off

Figure A.1: Measure Spatial Sematic Entropies, Additional Measure Configurations



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=on

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=off

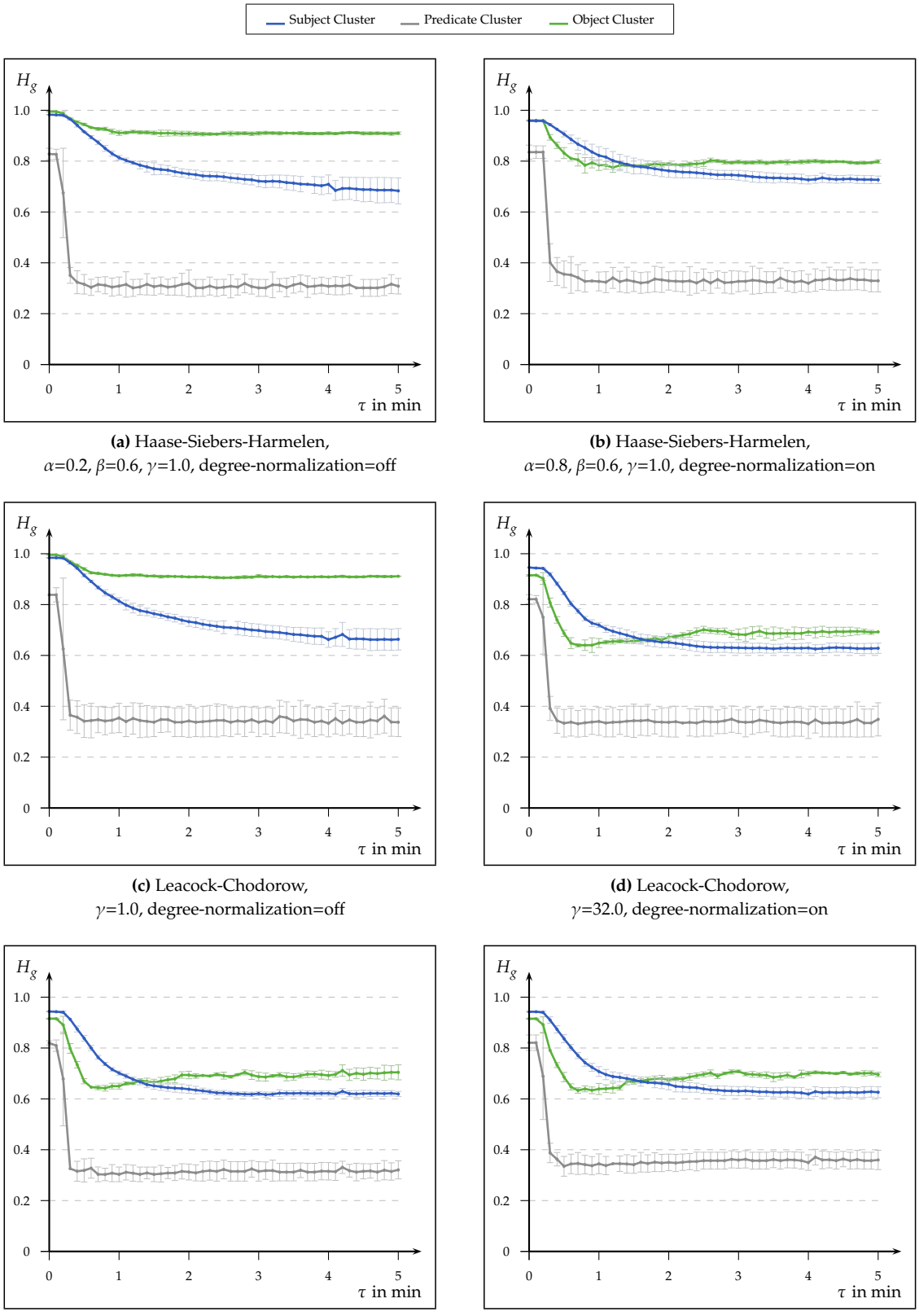
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=on

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=off

(e) Lin,
γ=1.0, degree-normalization=on

(f) Lin,
γ=3.0, degree-normalization=off

Figure A.2: Measure Spatial Sematic Entropy Gain, Additional Measure Configurations



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

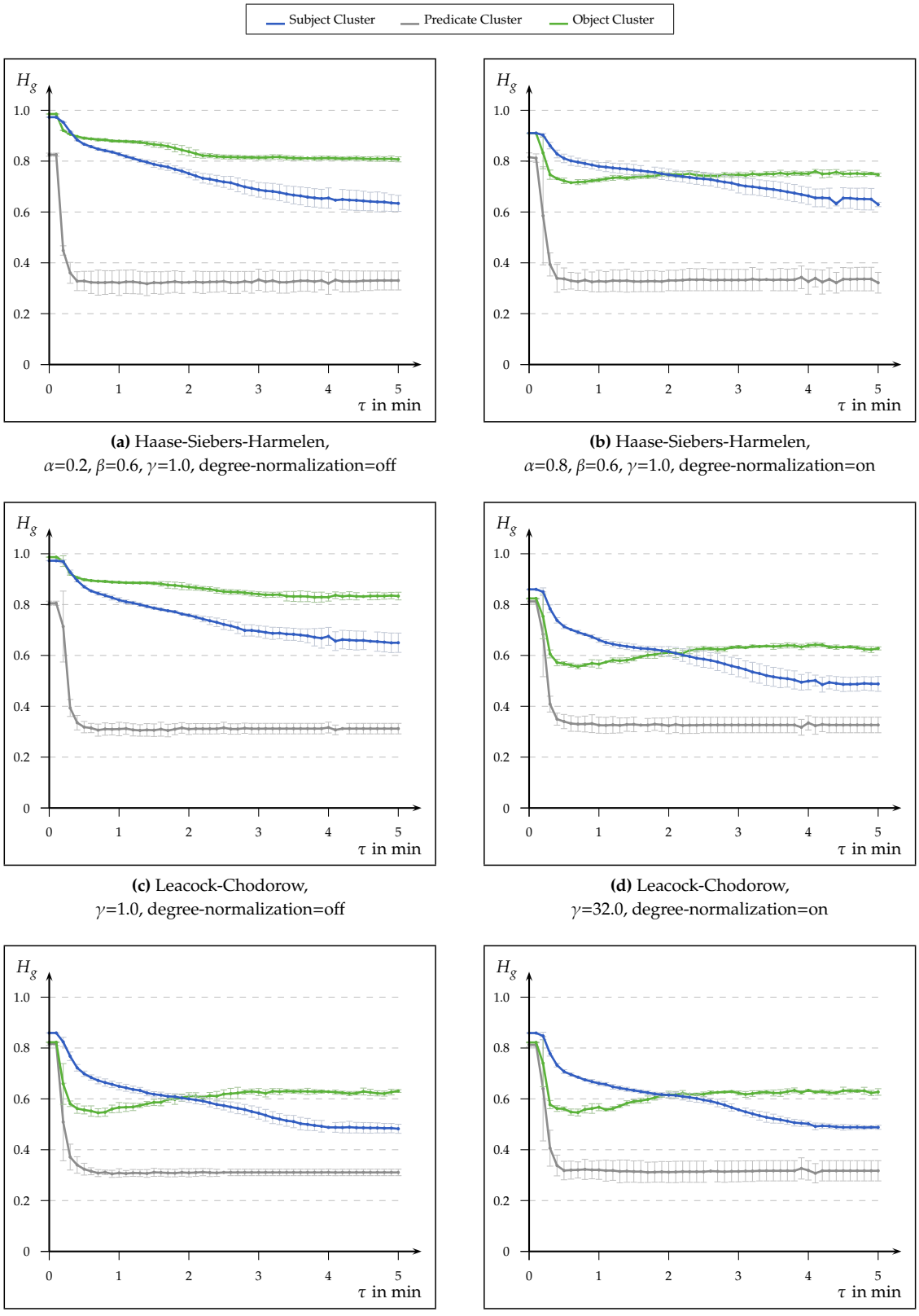
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure A.3: Measure Spatial Sematic Entropy Gain, Network 1, Ontology 1



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

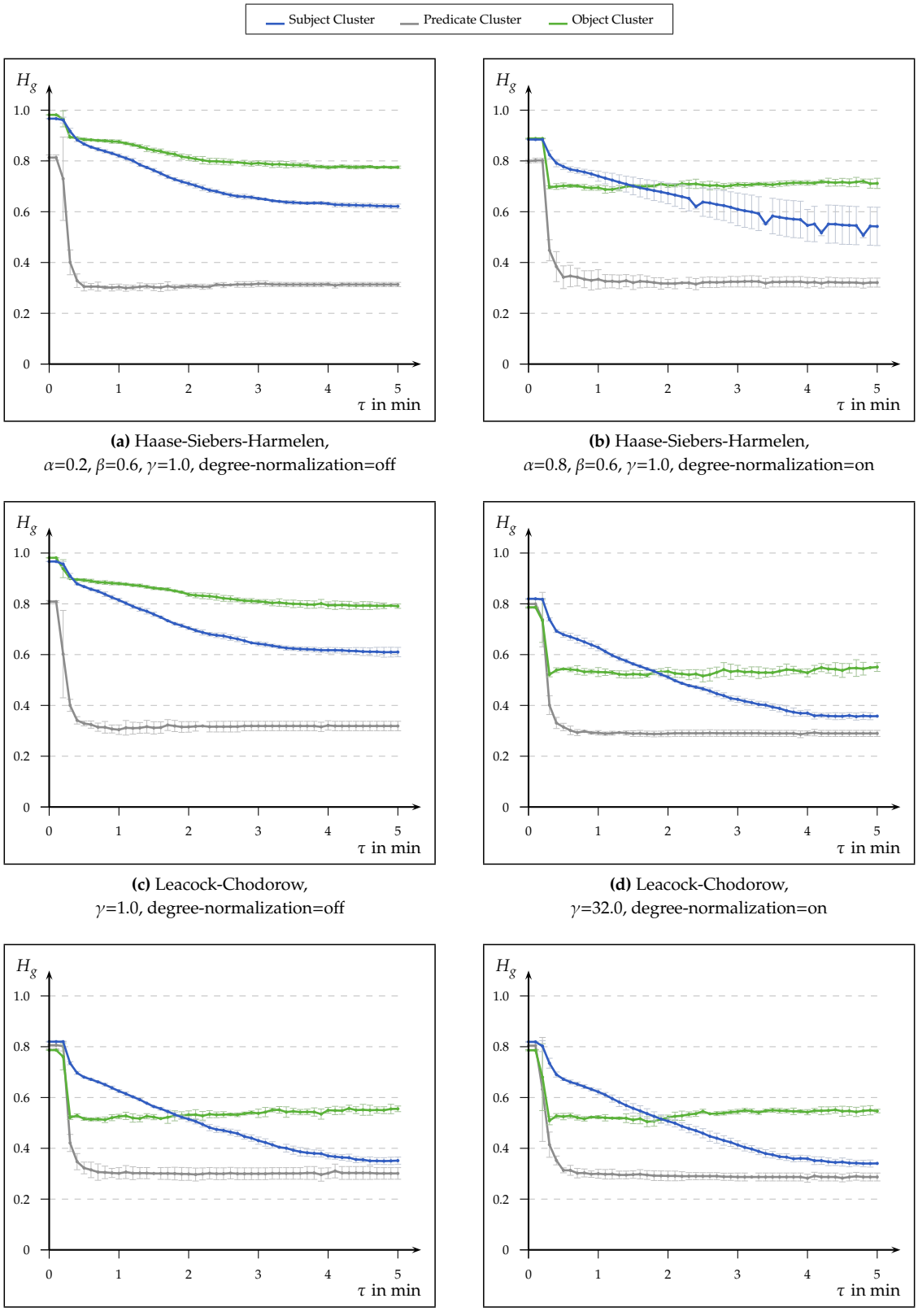
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure A.4: Measure Spatial Sematic Entropy Gain, Network 2, Ontology 1



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

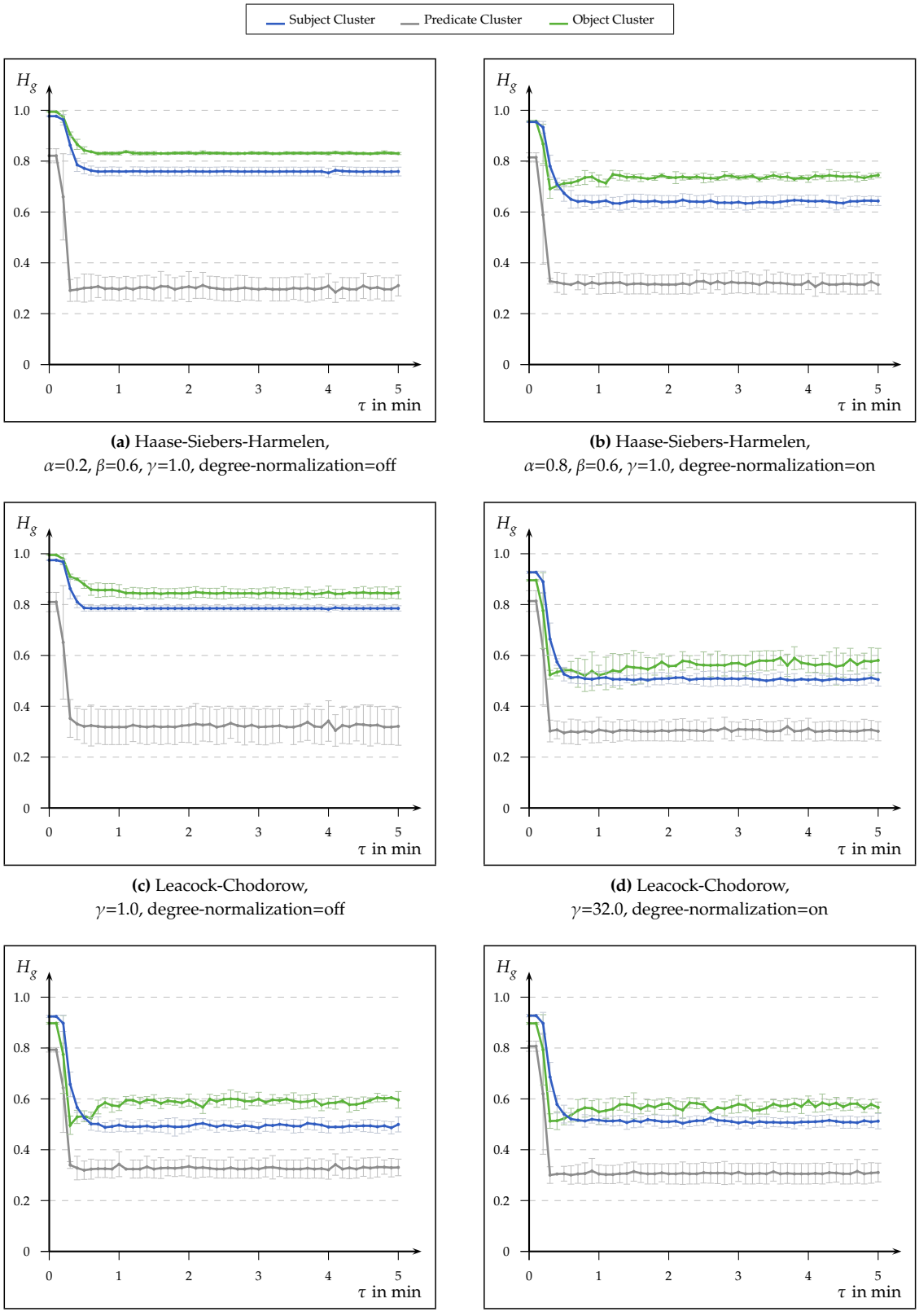
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure A.5: Measure Spatial Sematic Entropy Gain, Network 3, Ontology 1



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

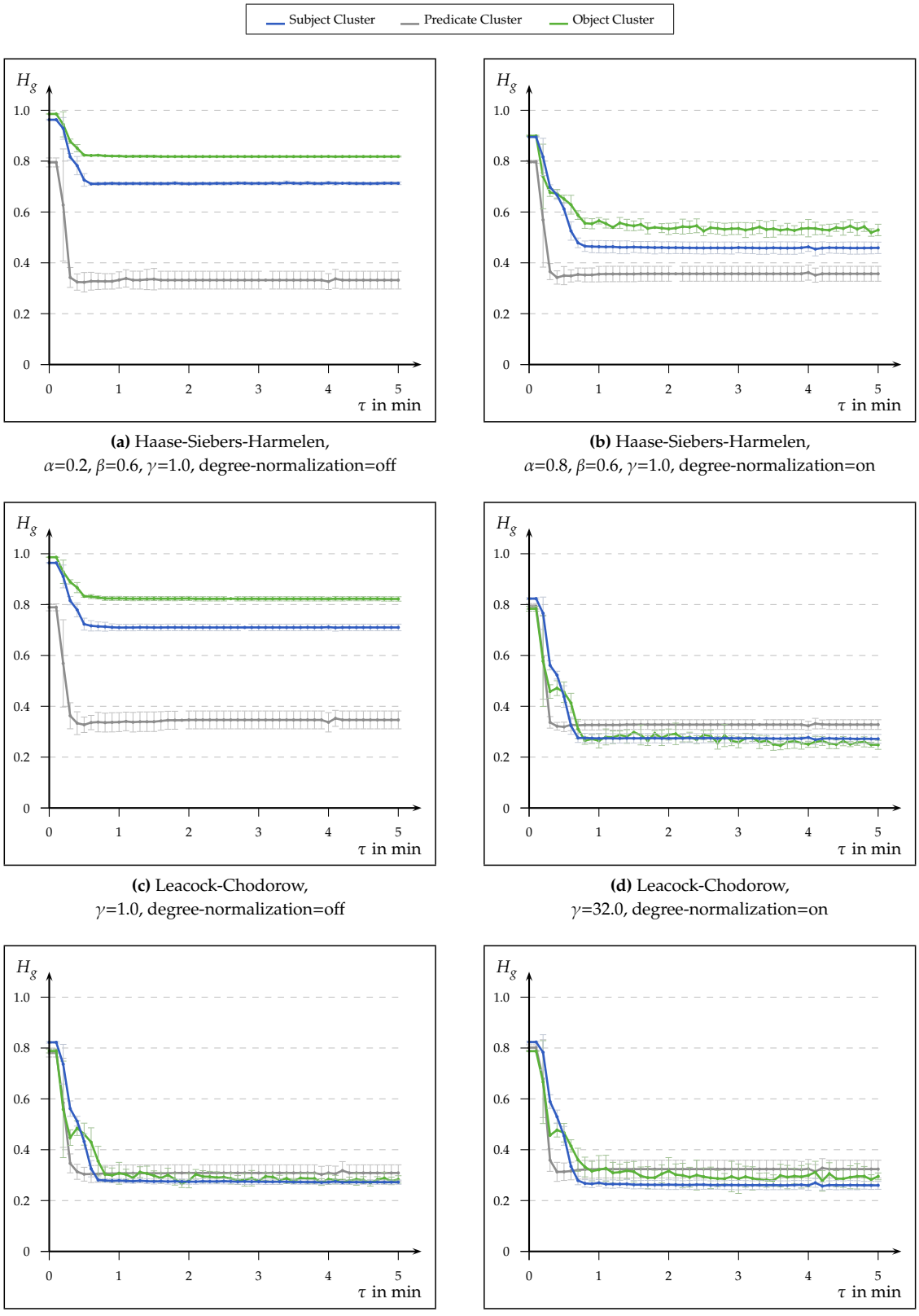
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure A.6: Measure Spatial Sematic Entropy Gain, Network 1, Ontology 2



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

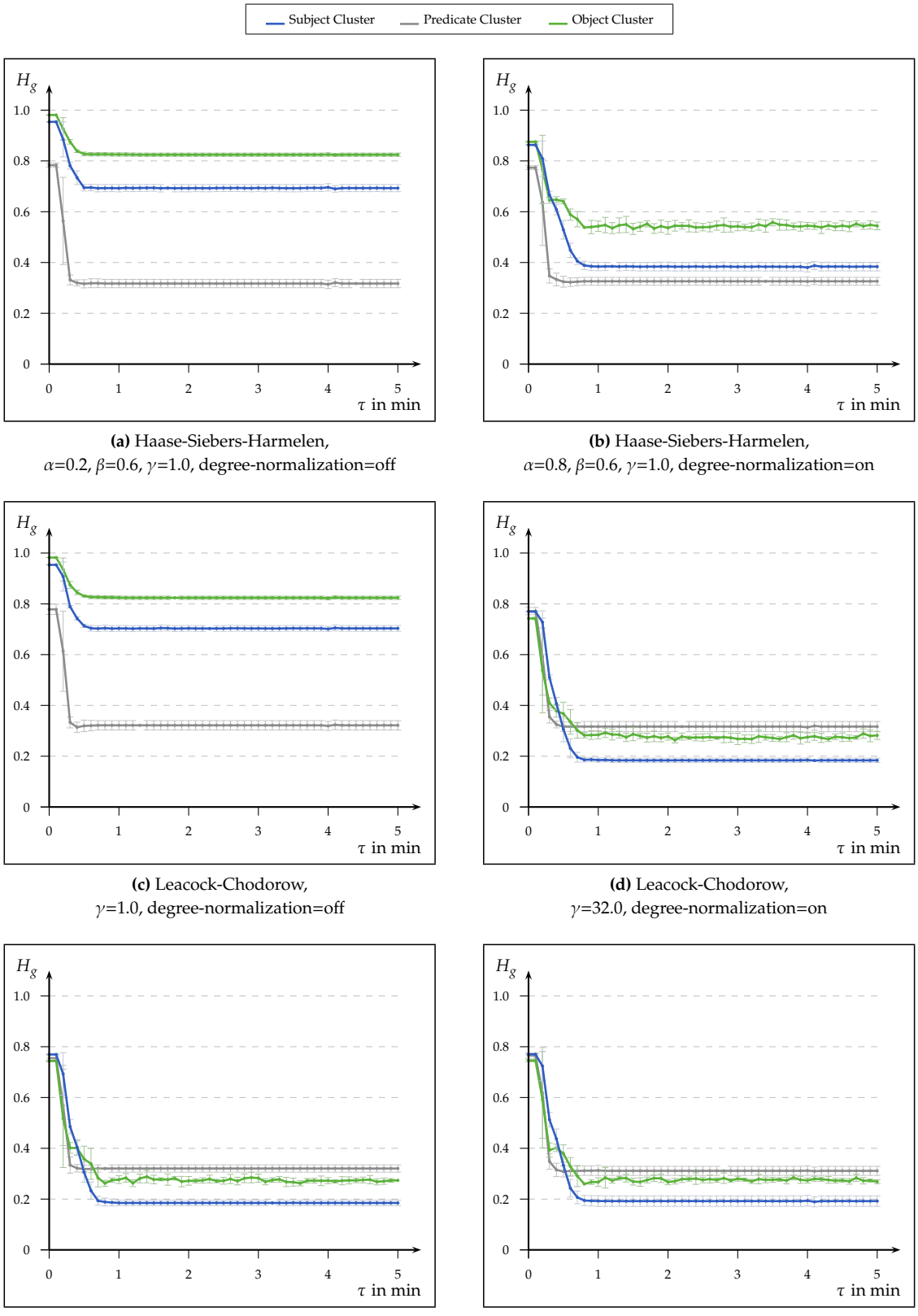
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure A.7: Measure Spatial Sematic Entropy Gain, Network 2, Ontology 2



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

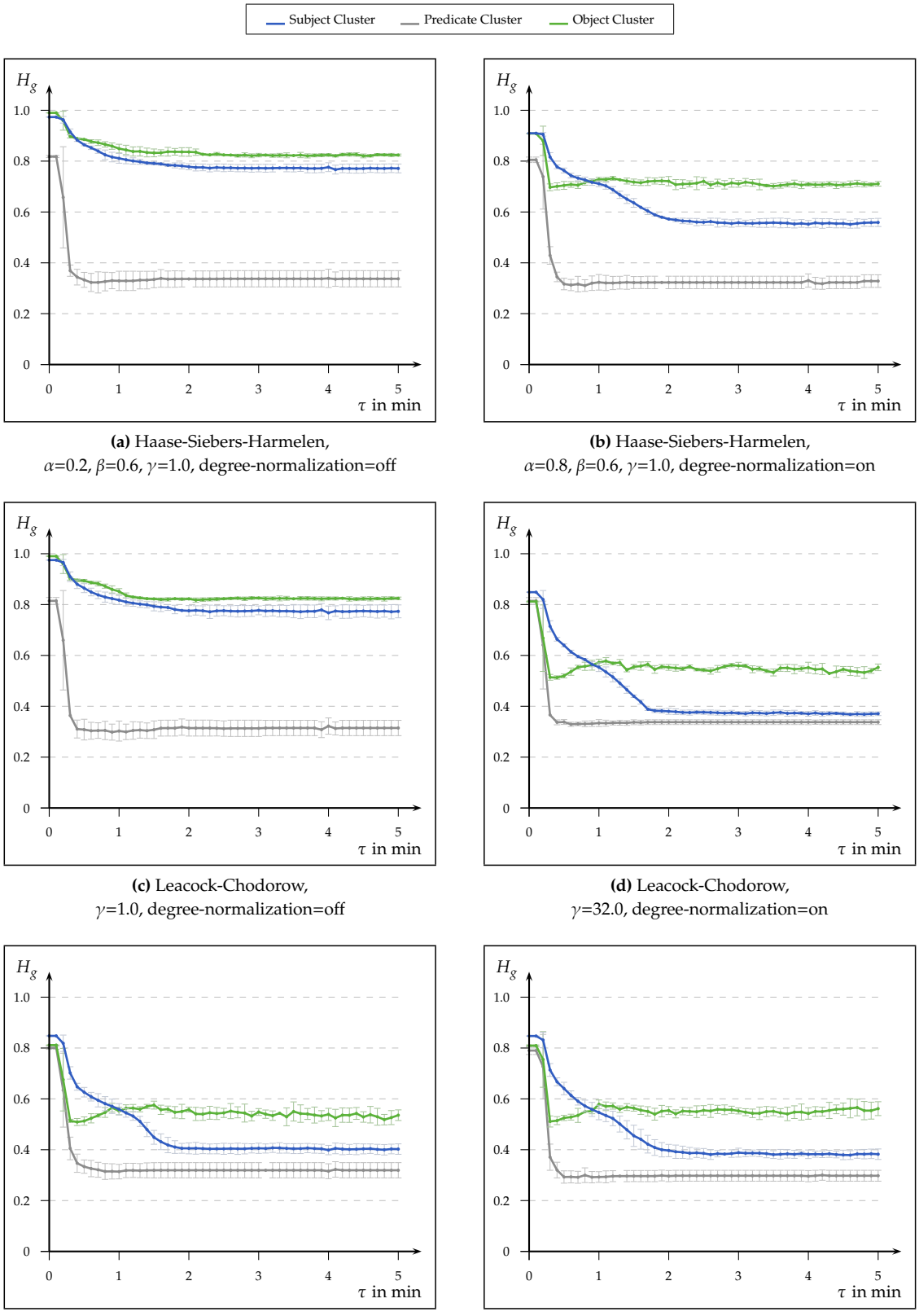
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure A.8: Measure Spatial Sematic Entropy Gain, Network 3, Ontology 2



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

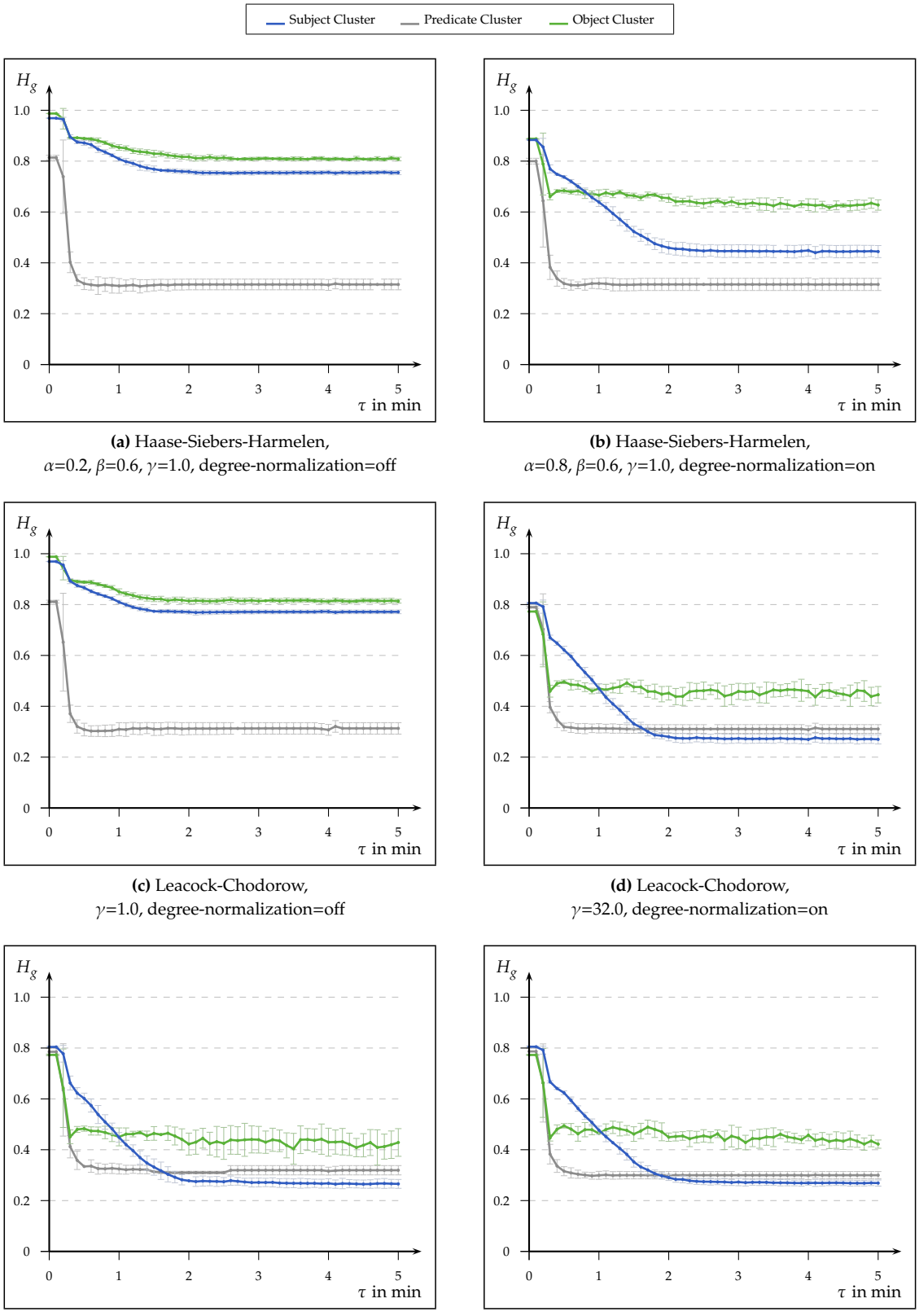
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure A.9: Measure Spatial Sematic Entropy Gain, Network 2, Ontology 3



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=off

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=on

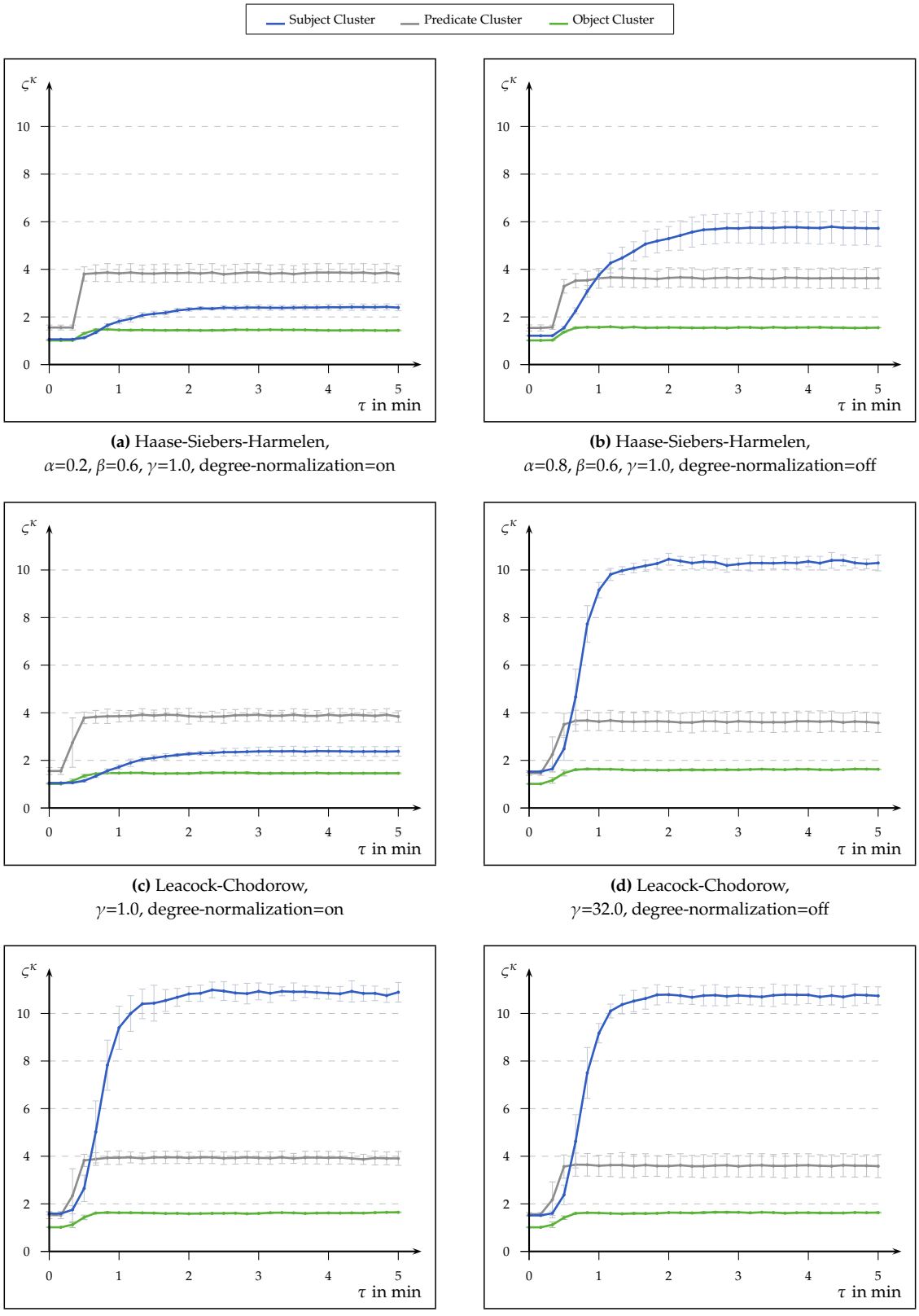
(c) Leacock-Chodorow,
γ=1.0, degree-normalization=off

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=on

(e) Lin,
γ=1.0, degree-normalization=off

(f) Lin,
γ=3.0, degree-normalization=on

Figure A.10: Measure Spatial Sematic Entropy Gain, Network 3, Ontology 3



(a) Haase-Siebers-Harmelen,
α=0.2, β=0.6, γ=1.0, degree-normalization=on

(b) Haase-Siebers-Harmelen,
α=0.8, β=0.6, γ=1.0, degree-normalization=off

(c) Leacock-Chodorow,
γ=1.0, degree-normalization=on

(d) Leacock-Chodorow,
γ=32.0, degree-normalization=off

(e) Lin,
γ=1.0, degree-normalization=on

(f) Lin,
γ=3.0, degree-normalization=off

Figure A.11: Measure Local similarity gain

Appendix B

Legal Notices

Netlogo Copyright Information

To reference this software in academic publications, please use: Wilensky, U. (1999). NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

The project gratefully acknowledges the support of the National Science Foundation (REPP and ROLE Programs) - grant numbers REC 9814682 and REC 0126227.

MersenneTwisterFast

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Colt

MRJ Adapter

Quaqua

JHotDraw

MovieEncoder

in this distribution fall under this license; the exceptions contain open source licenses embedded in the source code files themselves. In this license the Authors means the Copyright Holders listed above, and the license itself is Copyright 2003 by Sean Luke.

The Authors hereby grant you a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

to use, reproduce, modify, display, perform, sublicense and distribute all or any portion of the source code or binary form of this software or related data with or without modifications, or as part of a larger work; and under patents now or hereafter owned or controlled by the Authors, to make, have made, use and sell ("Utilize") all or any portion of the source code or binary form of this software or related data, but solely to the extent that any such patent is reasonably necessary to enable you to Utilize all or any portion of the source code or binary form of this software or related data, and not to any greater extent that may be necessary to Utilize further modifications or combinations.

In return you agree to the following conditions:

If you redistribute all or any portion of the source code of this software or related data, it must retain the above copyright notice and this license and disclaimer. If you redistribute all or any portion of this code in binary form, you must include the above copyright notice and this license and disclaimer in the documentation and/or other materials provided with the distribution, and must indicate the use of this software in a prominent, publically accessible location of the larger work. You must not use the Authors's names to endorse or promote products derived from this software without the specific prior written permission of the Authors.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS, NOR THEIR EMPLOYERS, NOR GEORGE MASON UNIVERSITY, BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

JpegImagesToMovie

© Copyright (c) 1999-2001 Sun Microsystems, Inc. All Rights Reserved.

Sun grants you ("Licensee") a non-exclusive, royalty free, license to use, modify and redistribute this software in source and binary code form, provided that i) this copyright notice and license appear on all copies of the software; and ii) Licensee does not utilize the software in a manner which is disparaging to Sun.

This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED

WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This software is not designed or intended for use in on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Licensee represents and warrants that it will not use or redistribute the Software for such purposes.

JOGL

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You acknowledge that this software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

Matrix3D

ASM

TRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Log4J, Collects15, Google Collections

Jena Copyright Information

GENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SimMetrics

Published under the GNU General Public License http://www.gnu.org/licenses/gpl.html.

JAMA

This software is a cooperative product of The MathWorks and the National Institute of Standards and Technology (NIST) which has been released to the public domain. Neither The MathWorks nor NIST assumes any responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic.