LarKC

The Large Knowledge Collider
a platform for large scale integrated reasoning and Web-search

FP7 – 215535

D1.4.1

Initial Framework for Measuring and Evaluating Heuristic Problem Solving

Zhisheng Huang (VUA), Annette ten Teije (VUA), Frank van Harmelen (VUA), Gaston Tagni (Coordinator, VUA), Hansjorg Neth (MPG), Lael Schooler (MPG), Sebastian Rudolph (AIFB, University of Karlsruhe), Pascal Hitzler (AIFB, University of Karlsruhe), Tuvshintur Tserendorj (FZI Karlsruhe), Yi Huang (Siemens), Danica Damljanovic, Angus Roberts (University of Sheffield)

Document Identifier:	LarKC/2008/D1.4.1
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	version 1.0.0
Date:	September 30, 2009
State:	final
Distribution:	public

Executive Summary

One of the key aspects in the development of LarKC is how to evaluate the performance of the platform and its constituent components in order to guarantee that the execution of a pipeline will match the user's needs and provide the desired solutions (answers) to the user's queries. Therefore, in this deliverable, the first in a series three documents concerned with the definition of a Framework for Measuring and Evaluating Heuristic Problem Solving, we make the first steps towards defining such framework by considering the theoretical foundations and principles of evaluation and measurement theory, discussing several important aspects related to the process of evaluating LarKC and its platform and, reporting on several dimensions and methods by which the components of the platform and the platform itself can be evaluated. Our primary interest in the context of this deliverable is a formative evaluation framework that is designed by the members of the LarKC project and aims to identify project strengths and weaknesses, focus managerial and research efforts, and measure progress towards achieving the project goals. Such a framework can also serve as the foundation for a subsequent summative evaluation.

Document Information

IST Project Number	FP7 – 215535	**Acronym**	LarKC
Full Title	Large Knowledge Collider		
Project URL	`http://www.larkc.eu/`		
Document URL			
EU Project Officer	Stefano Bertolo		

Deliverable	**Number**	1.4.1	**Title**	Initial Framework for Measuring and Evaluating Heuristic Problem Solving
Work Package	**Number**	1	**Title**	Conceptual Framework

Date of Delivery	**Contractual**	M18	**Actual**	30-September-09
Status	version 1.0.0		final ☒	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Gaston Tagni, Zhisheng Huang, Frank van Harmelen, Annette ten Teije (VUA), Hansjörg Neth, Lael J. Schooler (MPG), Sebastian Rudolph, Pascal Hitzler (AIFB, University of Karlsruhe), Tuvshintur Tserendorj (FZI Karlsruhe), Yi Huang (Siemens), Danica Damljanovic, Angus Roberts (University of Sheffield)			
Resp. Author	Gaston Tagni		**E-mail**	gtagni@cs.vu.nl
	Partner	VUA, MPG, Siemens, University of Sheffield	**Phone**	+31 20 59 87753

Abstract (for dissemination)	One of the key aspects in the development of LarKC is how to evaluate the performance of the platform and its constituent components in order to guarantee that the execution of a pipeline will match the user's needs and provide the desired solutions (answers) to the user's queries. Therefore, in this deliverable, the first in a series three documents concerned with the definition of a Framework for Measuring and Evaluating Heuristic Problem Solving, we make the first steps towards defining such framework by considering the theoretical foundations and principles of evaluation and measurement theory, discussing several important aspects related to the process of evaluating LarKC and its platform and, reporting on several dimensions and methods by which the components of the platform and the platform itself can be evaluated. Our primary interest in the context of this deliverable is a formative evaluation framework that is designed by the members of the LarKC project and aims to identify project strengths and weaknesses, focus managerial and research efforts, and measure progress towards achieving the project goals. Such a framework can also serve as the foundation for a subsequent summative evaluation.
Keywords	Heuristic Problem Solving, Evaluation Framework, Measurement, Evaluation, Formative Evaluation

Project Consortium Information

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck `http://www.sti-innsbruck.at`		Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
AstraZeneca AB `http://www.astrazeneca.com/`		Bosse Andersson AstraZeneca Lund, Sweden E-mail: bo.h.andersson@astrazeneca.com
CEFRIEL SCRL. `http://www.cefriel.it/`		Emanuele Della Valle CEFRIEL SCRL. Milano, Italy E-mail: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O. `http://cyceurope.com/`		Dr. Michael Witbrock CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia E-mail: witbrock@cyc.com
Hchstleistungsrechenzentrum, Universitaet Stuttgart `http://www.hlrs.de/`		Georgina Gallizo Hchstleistungsrechenzentrum, Universitaet Stuttgart Stuttgart, Germany E-mail : gallizo@hlrs.de
Max-Planck-Institut fur Bildungsforschung `http://www.mpib-berlin.mpg.de/index_js.en.htm`		Dr. Lael Schooler, Max-Planck-Institut fr Bildungsforschung Berlin, Germany E-mail: schooler@mpib-berlin.mpg.de
Ontotext Lab, Sirma Group Corp. `http://www.ontotext.com/`		Atanas Kiryakov, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: atanas.kiryakov@sirma.bg
SALTLUX INC. `http://www.saltlux.com/EN/main.asp`		Kono Kim SALTLUX INC Seoul, Korea E-mail: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT `http://www.siemens.de/`		Dr. Volker Tresp SIEMENS AKTIENGESELLSCHAFT Muenchen, Germany E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD `http://www.shef.ac.uk/`		Prof. Dr. Hamish Cunningham THE UNIVERSITY OF SHEFFIELD Sheffield, UK E-mail: h.cunningham@dcs.shef.ac.uk
VRIJE UNIVERSITEIT AMSTERDAM `http://www.vu.nl/`		Prof. Dr. Frank van Harmelen VRIJE UNIVERSITEIT AMSTERDAM Amsterdam, Netherlands E-mail: Frank.van.Harmelen@cs.vu.nl
THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY `http://www.iwici.org/`		Prof. Dr. Ning Zhong THE INTERNATIONAL WIC INSTITUTE Mabeshi, Japan E-mail: zhong@maebashi-it.ac.jp
INTERNATIONAL AGENCY FOR RESEARCH ON CANCER `http://www.iarc.fr/`		Dr. Paul Brennan INTERNATIONAL AGENCY FOR RESEARCH ON CANCER Lyon, France E-mail: brennan@iarc.fr

Table of Contents

List of Abbreviations

TTB	Take the Best
RDF	Resource Description Framework

1 Introduction

The LarKC project is concerned with large scale semantic reasoning and more specifically aims at providing a platform for massive, distributed and incomplete reasoning that will remove the scalability barriers imposed by current Semantic Web reasoning systems. Such a platform is realized through a pluggable architecture consisting of several plug-in types: Identify, Select, Transform, Reason and Decide plug-ins, each of which is responsible for solving a specific type of (sub) problem. For an overview of the platform and the initial operational framework the reader is referred to Deliverable 1.2.1.

One way to describe the LarKC platform is to think of it as a problem-solving engine that employs multiple techniques and methods to provide solutions, in the form of answers to SPARQL-like queries, to problems (queries) initiated by users and applications. According to this view, solutions to a problem are computed by executing a workflow whose activities perform specific tasks, i.e. solve specific (sub) problems, and contribute in this way to finding a solution to the problem at hand. Each activity is carried out by a plug-in type and each plug-in may use different problem-solving strategies to achieve its goal, some of which may depend on the type of plug-in, e.g. the use of standard methods used in the area of information retrieval for implementing selection plug-ins.

From this point of view, one of the key aspects in the development of LarKC is how to evaluate the performance of the platform and its constituent components in order to guarantee that the execution of a workflow will match the user's needs and provide the desired solutions (answers) to the user's queries. Therefore, in this deliverable, the first in a series three documents concerned with the definition of a Framework for Measuring and Evaluating Heuristic Problem Solving, we make the first steps towards defining such framework by considering the theoretical foundations and principles of evaluation and measurement theory, discussing several important aspects related to the process of evaluating LarKC and its platform and, reporting on several dimensions and methods by which the components of the platform and the platform itself can be evaluated. Our primary interest in the context of this deliverable is a formative evaluation framework that is designed by the members of the LarKC project and aims to identify project strengths and weaknesses, focus managerial and research efforts, and measure progress towards achieving the project goals. Such a framework can also serve as the foundation for a subsequent summative evaluation.

Structure of the document. This document is structured as follows: in Chapter 2 we provide a clarification of the deliverable's main concepts and an overview over the specific challenges presented by the evaluation of the LarKC project and platform. This chapter clarifies the notions of *measurement*, *evaluation* and *heuristic problem solving*—both in general and in the specific context of the LarKC project. In this chapter we also identify and discuss an initial set of quantitative dimensions that we think need to be measured in order to evaluate LarKC. Such metrics include robustness, speed, accuracy, scalability, usability and user satisfaction and quality and impact. After introducing the basic notions and principles of evaluation theory, chapters 3, 4 and 5 report on a series of techniques and quantitative dimensions or criteria that can be used for the purposes of evaluating identify, transform and reason plug-in types. More specifically, Chapter 3 includes a specification of different criteria and techniques for evaluating selection and retrieval plug-in types. In the same way, in Chapter 4 we explore a series of evaluation criteria and measures that can be utilized to evaluate the quality of the transformation plug-ins within the LarKC platform and then, in Chapter 5 we lay the foundations for a statistical approach to evaluating approximate and anytime reasoning algorithms as they play a prominent role in achieving and implementing heuristic problem solving methods. Finally, Chapter 6 reports the conclusions.

2 Theory of Evaluation

The purpose of this chapter is to provide a clarification of the deliverable's main concepts and an overview over the specific challenges presented by the evaluation of the LarKC project and platform (Fensel et al., 2008). As this deliverable is entitled "Initial framework for measuring and evaluating heuristic problem solving" we will first clarify the notions of *measurement*, *evaluation* and *heuristic problem solving*—both in general (Section 2.1) and in the specific context of the LarKC project (Section 2.2).

2.1 Conceptual Clarifications

2.1.1 Measurement and Evaluation

As the notions of measurement and evaluation are closely related we will treat them in the same section.

Measurement is defined as the act or process of assigning numbers to phenomena according to a rule (according to WordNet at `http://wordnetweb.princeton.edu`), as well as the result of such an act or process. Obtaining the magnitude of a quantity, such as the duration of a time interval or the mass of an object, presupposes a unit of measurement, such as a second or a kilogram. Consequently, a crucial part of measuring something consists in designing and choosing the correct metrics and appropriate standards.

When measurement provides the basis for a judgment concerning the value, merit or significance of some process or project we speak of *evaluation*. Thus, evaluation contains measurement as a critical part: Both measuring and evaluating require specific standards or criteria to quantify some dimension, but whereas measurement is neutral evaluation contains an additional attribution of value that relates the level of observed achievement to pre-defined objectives.

Generally speaking, the purpose of evaluation is to provide answers to questions like 'Is it any good?', 'What is it good for?' and 'How good is it *really*, i.e., relative to its original goals and compared to available alternatives?' The common element of all these questions is the attribution of value, whereas the questions get progressively more specific by implying practical uses and particular goals and constraints. The generic "it" in all questions can refer to a particular thing, method or class of methods, a product, policy or paradigm, or—as in the case of LarKC—a multi-year research effort.

More formally, *evaluation* is the systematic determination of the value, merit or significance of a practice, process or project in relation to explicitly defined standards. The process of evaluation typically serves multiple *functions* (Stockmann, 2000):

1. *Knowledge*: An explicit definition of standards and the collection of quantitative and qualitative data generates knowledge and enables new insights about the degree to which the project or platform provides the desired functionality.

2. *Control*: Additional knowledge enables the allocation of management efforts, the discovery and correction of deficits, as well as the identification of strengths.

3. *Communication*: Enable a continuous dialog between project partners, potential and actual user communities, various stakeholders and the public.

4. *Legitimization*: Provide a rationale why the original investments (in terms of research efforts and monetary funds) were justified and help to allocate future resources.

An important distinction in evaluation theory is that between summative and formative evaluations (Scriven, 1991). Although evaluations are always intended as a basis for both assessment and improvement, some focus more on the assessment aspect whereas others aim more towards continuous adjustments and improvement:

- *summative evaluation* provides information on the overall efficacy or efficiency of a product or process. Conducting a summative evaluation requires identifying large-scale patterns in performance and comparing quantitative and qualitative measurements to pre-defined standards in order to obtain an overall judgment. Summative evaluations are often carried out by an external authority and aim to gain closure and a retrospective appraisal of a project's merit and achievements. By contrast,

- *formative evaluation* is a process of ongoing feedback on performance. The distinctive features of this type of evaluation are to identify aspects of performance that need to improve and to offer corrective suggestions while the project is still in progress. Formative evaluations are often carried out internally (i.e., by the partners that design and collaborate on a project), rely on multiple iterative cycles in which performance assessments are compared to pre-defined objectives and aim to approach the desired standards in multiple steps. Consequently, any formative evaluation is closely related to the process of *benchmarking* in process management and software engineering.

An informal way of contrasting both subtypes of evaluation is attributed to Robert Stakes (according to Scriven, 1991, p. 69): "When the cook tastes the soup, that's formative; when the guests taste the soup, that's summative." Table 2.1 summarizes the main features of both types of evaluation on several dimensions.

The dichotomous distinction between formative and summative evaluations has been criticized as being too limited, particularly with respect to projects that generate new knowledge or develop new systems or interventions (Patton, 1996). Rather than trying to categorize an evaluation effort as being of a particular type we believe that the value of the distinction between summative and formative evaluations consists in providing useful dimensions that allow to plan, design and improve measurements and the evaluation process.

A related dichotomy is the distinction between intrinsic and extrinsic evaluations. The distinction presupposes that the evaluated component is part of a greater whole and assumes two different perspectives. Whereas an *intrinsic evaluation* assesses the performance of a single component in isolation (and assumes

	Type of evaluation:	
Dimension	*summative*:	*formative*:
main function:	assessment	improvement
evaluator:	external: users or sponsors	internal: researchers, developers and program managers
evaluatee:	product (running system)	process (system development and improvement)
time perspective:	retrospective	concurrent
frequency:	once, upon completion	multiple iterative cycles

Table 2.1: Basic features of summative vs. formative evaluations.

that all external factors are held constant) an *extrinsic evaluation* views a component as part of a larger system and assesses the component with regards to its overall functionality.

2.1.2 Heuristic Problem Solving

The object or process to be evaluated according to this deliverable's title is "heuristic problem solving". This raises the questions: What is meant by "heuristics" and why or when is the use of heuristics indicated or successful?

What defines a *heuristic*? Based on the Greek term for "serving to find out or discover" heuristics have been invoked in so many meanings and contexts that some researchers (e.g., Shah & Oppenheimer, 2008) claim that the word has become arbitrary and lost its meaning. Although it is true that the notion of heuristics has experienced a mixed history (see Gigerenzer, Todd, & the ABC research group, 1999; Gigerenzer & Brighton, 2009, for overviews) and has been used differently in different disciplines (e.g., in biology, computer science, engineering, physics, and psychology) we think that heuristics have often been misunderstood or underestimated and deserve to be rehabilitated.

On the most general level, the notion of a heuristic is used almost interchangeably with "method" and refers to a procedure that aims to increase the probability of solving some problem. On this vague level, heuristics may be simply rules-of-thumb, intuitive judgments or educated guesses that use experience-based techniques for problem solving, learning and discovery.

In the context of mathematical and computational problems heuristics are often invoked when an optimal solution is either practically infeasible or principally unknown. For many problems for which an optimal solution is either computationally intractable or cannot be proven to be correct heuristic methods (such as hill-climbing, simulated annealing, or tabu search) provide approximate solutions that are good enough for practical purposes (e.g., in terms of speed, required accuracy and computational efforts). Many such heuristics can be shown to suffer from specific weaknesses (like the susceptibility of hill-climbing to get stuck on local maxima) or to approximate an optimal solution, given sufficient time and computational power (like Q-learning algorithms in machine learning).

Interestingly, one of the standard textbooks on *Modern heuristics* (Michalewicz & Fogel, 2004)—an updated edition of Polya's classic text *How to solve it*, 1947)—does not even define the meaning of heuristics, but uses the term generically to describe any method that is suited to solve a given problem. Practically, the book describes ways to analyze problems and an arsenal of algorithms that yield acceptable solutions with limited computational resources.

A common thread in these treatments is that heuristics may yield satisfactory solutions, but are ultimately only second-best strategies for cases for which an optimization method is lacking. This assumption of an inferior nature of heuristics is even retained by explicit advocates of heuristics. For instance, Zanakis and Evans (1981) blur the terminological boundaries by recommending *heuristic "optimization"* and state that the use of heuristics is "desirable and advantageous" (p. 85) under the following circumstances:

1. Inexact or limited data contains errors larger than the "sub-optimality" of a good heuristic.
2. A simplified model (or inaccurate representation) is used.
3. A reliable exact method is not available.
4. An exact method is available, but it is computationally infeasible.
5. To improve the performance of an optimizer by providing initial solutions and narrowing down the search space.
6. There is a repeated need to solve the same problem, making speed more important than accuracy.
7. A heuristic solution is satisfactory.
8. Heuristics are simple and understandable, so that users are more likely to adopt a method.
9. Heuristics serve as a learning device to gain new insights.
10. Other resource limitations (e.g., of time or budget) enforce the use of heuristics.

Clearly, heuristics in this sense are mainly defined by the absence of some desirable state (better data or models, more time, larger budget, etc.), are merely indicated when no superior techniques are available and yield only second-best solutions when compared to the results of optimization techniques. Despite their uses, heuristics yield merely quick-and-dirty solutions for pragmatically minded people.

Given this heritage, it may not be surprising that a tradition within cognitive psychology has attributed irrational elements of human judgment to the use of heuristics. Just as many computer scientists and engineers viewed heuristics as computational shortcuts that yield suboptimal solutions the *heuristics-and-biases* framework championed by Tversky and Kahneman (1974) has nurtured the impression that the use of heuristics by naive humans is responsible for many erroneous or irrational decisions.

In contrast to these negative views of heuristics, the Center for Adaptive Behavior and Cognition of Berlin's Max-Planck Institute for Human Development defines heuristics as effective and efficient solution methods that ignore information. Unlike optimization methods, which are attempting to maximize some criterion, heuristics aim to *satisfice*, i.e., they choose the first option that exceeds an aspiration level. This definition preserves key elements of Herbert Simon's characterization of methods of heuristic search as examples of rational adaptation and as "methods for arriving at satisfactory solutions with modest amounts of computation" (Simon, 1990, p. 11) and emphasizes the positive potential of heuristics to guide information search and modify problem representations to facilitate solutions.

It would be premature to interpret the use of terms like "satisfactory" and "modest" as a confirmation of the aforementioned skepticisms against the use of heuristics. Numerous studies of naturalistic decision scenarios and simulated environments have shown that heuristics neither need to be poor surrogates for optimization methods nor merely yield second-best solutions (see Gigerenzer et al., 1999; Gigerenzer, 2000; Gigerenzer & Selten, 2001; Gigerenzer, 2008, for overviews).

An example of a simple heuristic that demonstrably outperforms many more elaborate strategies is the Take-The-Best (TTB) heuristic introduced by Gigerenzer and Goldstein (1996). When two objects (e.g., cities) are compared on some dimension (e.g., population) and both objects are recognized and characterized by various cues (e.g., the presence of a university, a major league soccer team, an exposition site, etc.) TTB describes a process to choose one of the options. Assuming that cues are inspected sequentially in order of decreasing validity (defined as the probability of a correct response based on the cue) TTB makes a choice based on the first cue that discriminates between both objects. For instance, when asked to determine whether Hamburg or Heidelberg is bigger TTB would first consider the most valid cue, say, the presence of a university. As both cities have a university the search inspects the next most valid cue, e.g., the soccer team cue. As only Hamburg has a major league team it will be chosen as the more populous city.

TTB ignores information as cues with less predictive validity then the first discriminating one (e.g., the presence of an exposition site) are not even considered. Furthermore, TTB is non-compensatory as potentially conflicting information on multiple cues is never integrated. Despite its fast and frugal nature (by essentially betting on one good reason) TTB frequently outperforms much more elaborate decision strategies like multiple regression (see Czerlinski, Gigerenzer, & Goldstein, 1999; Goldstein & Gigerenzer, 2002; Martignon & Hoffrage, 1999, 2002, for analyses).

One reason for the success of TTB is that it is adapted to stable structures in the environment (e.g., the fact that bigger cities tend to have universities and soccer teams) and enabled by evolved psychological capacities (e.g., recognition memory for cities and cues that are regularly mentioned in the media). Such an intimate match between environmental structure and human capacities is an instance of *ecological rationality* (Gigerenzer et al., 1999). A second reason for the remarkable success of simple heuristics is that more information or computation does not necessarily yield better results. Although a more complex theoretical model can achieve a better fit to an existing data set, using less information can yield more robust predictions when dealing with fundamentally uncertain systems (e.g., forecasting the weather or stock market trends).

Our example of TTB also dispels the prejudice that heuristics are merely vague rules of thumbs or malleable labels that can be accommodated to any kind of post-hoc explanation. While this criticism may apply to so-called heuristics like 'availability' or 'representativeness' (which are really just verbal labels) TTB has been formulated as an explicit and precise process model that contains three building blocks:

1. a *search rule*: Consider cues in order of decreasing validity,
2. a *stopping rule*: Terminate search at the first cue that discriminates between the available options, and a
3. *decision rule*: Choose the option with the higher criterion value on the discriminating cue.

Consequently, TTB can be formally analyzed, computationally implemented and subjected to rigorous tests that compare and contrast its predictions with those of alternative strategies (see Czerlinski et al., 1999; Gigerenzer & Brighton, 2009, for such tests).

In summary, simple yet precise heuristics that are ecologically rational can be powerful methods and indispensable cognitive processes on par with logic and probability theory (Polya, 1947; Gigerenzer, 2008). Rather than being mere crutches and kluges, heuristics can be viewed as adaptive tools that are tailored to specific classes of problems and can outperform optimization strategies with respect to computational efforts and performance outcomes. A view of mind as an adaptive toolbox (as proposed by Gigerenzer et al., 1999; Gigerenzer & Selten, 2001) demonstrates how heuristics can successfully exploit evolved mental capacities and environmental structure and can succeed without trading off effectiveness against efficiency. Given this existence proof of homo heuristicus (Gigerenzer & Brighton, 2009) we need to ask to what extent heuristics can contribute fruitfully to the engineering challenges of the LarKC project.

2.2 Applications to LarKC

Whereas the previous section defined and described the concepts of measurement, evaluation and heuristic problem solving in general terms this section addresses the question how these notions can or need to be instantiated in the specific context of the LarKC project.

2.2.1 Measurement and Evaluation of LarKC

Measuring the performance of LarKC will be challenging. As we have seen in Section 2.1.1 any act measurement requires appropriate metrics and standards of measurement. Prior to any technical discussion regarding the instrumentation of the LarKC platform and its components it needs to be determined which quantities need to be measured on which criteria in order to determine its degree of success. The difficulties are aggravated by the fact that LarKC is a complex entity, consisting of multiple sub-components that need to interact with each other to produce a result. Identifying the source of a particular success or failure thus faces a credit assignment problem.

Whereas some quantities and criteria may be fairly general (like speed, accuracy, robustness) others may only apply to certain sub-components (e.g., the level of recall of a selector or soundness of a reasoner). Furthermore, it is conceivable that a particular component works perfectly well in the context of some other components, but fails or performs sub-optimally when used with different source data or in combination with different components. Before we can devise ways of dealing with this complexity we need to address the question: What are the main objectives of the LarKC project and how can progress towards these objectives be evaluated?

In contrast to the simple dichotomous view exemplified by Table 2.1 (see Page 6) a project of the magnitude of LarKC requires a hybrid and multi-layered type of evaluation. On a very general level any EU-sponsored project implements some mechanisms that bear the characteristics of summative evaluations (e.g., a thorough review and approval process by subject-matter experts external to the project) but also elements of formative evaluations (e.g., the internal quality review of deliverables by colleagues, partners of other work packages and the project management). Although such elements of evaluation that are already built into the management and administration of the LarKC project regularly take stock of the project's current status, provide important pointers to needs and weaknesses and thus shape the future direction of the project, they do not yet constitute an evaluative effort that is suited to determine the success or failure of LarKC.

The main goal of the current document and one of the missions of work package 1 (WP 1) is to develop a framework for the measurement and evaluation of the LarKC platform and its elements of heuristic problem solving. As this constitutes an internal endeavor that primarily focuses on iterative improvement this evaluation is mainly of a formative nature, but an apparatus to identify strengths and weaknesses and measure various performance aspects will eventually facilitate the summative evaluation of the project as well.

As has been stated in Section 2.1.1 any evaluative judgment on a project must be based on a project's original goals. In the context of the LarKC project, its degree of success needs to be evaluated in relation to the project's specific constraints and criteria. Specifically, judging the success and effectiveness of LarKC necessitates the clarification of three components (Figure 2.1):

1. *objectives*, i.e., an explicit definition of the project goals;
2. *metrics*, i.e., measuring the degree of achievement of these objectives on some quantifiable criteria; as well as an
3. *assessment of value*, i.e., some integrative judgment the extent to which its goals have been achieved or, alternatively, which goals have been achieved to what extent.

We will consider each of these points in turn to address the question how LarKC can be evaluated.

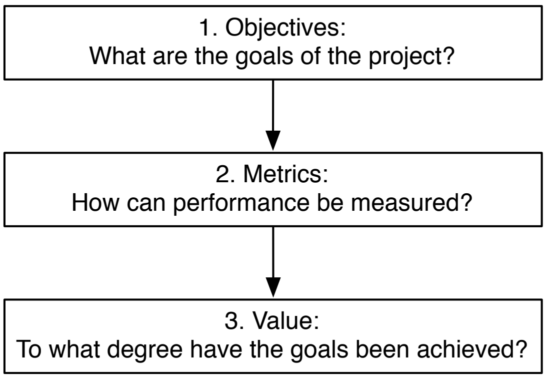

1. Objectives:
What are the goals of the project?
2. Metrics:
How can performance be measured?
3. Value:
To what degree have the goals been achieved?

Figure 2.1: Basic steps of the evaluation process.

Objectives of LarKC

Given the scope of the LarKC project it is no surprise that its goals are complex and multi-faceted.

As an European Union Framework 7 program that aims to provide a technological infra-structure and combines a creative vision with innovative engineering to push beyond what is currently possible LarKC shares many general goals with similar projects. Such general goals include providing an intellectual framework that allows the application of semantic technologies to real-world problems and creating the technical infrastructure that enables the desired functionality. The characteristics of this infrastructure can be measured by parameters—such as speed, accuracy and cost-effectiveness (in terms of the ratio between the quality of an answer to a query and the computational resources invested)—and will partly depend on the state-of-the-art of current hardware and software resources.

Beyond these general factors an evaluation of the LarKC platform will need to take into account LarKC-specific goals. According to its vision, LarKC is poised to provide a platform for massive distributed incomplete reasoning that removes the scalability barriers of existing reasoning systems for the semantic web. In contrast to existing semantic web applications LarKC aims to move beyond the current state-of-the-art of webscale reasoning. By developing new methods of data selection, transformation and semantic reasoning, employing cognitively inspired approaches and building a distributed platform that uses both high-performance computing clusters and a network of interlinked machines LarKC promises a level of scalability beyond what has been possible so far. As such a platform can hardly be developed in a contextual vacuum LarKC's leadership has included designated use-cases (dealing with the topics of urban computing, early clinical development and the production of carcinogenesis reference materials) that allow to assess and demonstrate the project's potential through practical applications.

According to Fensel et al. (2008, p. 2) the major objectives of LarKC are:

1. Design an integrated pluggable platform for large-scale semantic computing.
2. Construct a reference implementation for such an integrated platform for large-scale semantic computing, including a fully functional set of baseline plug-ins.

3. Achieve sufficient conceptual integration between approaches of heterogeneous fields (logical inference, databases, machine learning, cognitive science) to enable the seamless integration of components based on methods from these diverse fields.

4. Demonstrate the effectiveness of the reference implementation through applications in services that require data-aggregation (e.g. in the area of urban computing), meta-analysis of scientific literature in cancer research, and data-integration and -analysis in early clinical development and the drug-discovery workflow.

In addition to the requirements dictated by the questions addressed in the use cases two aspects that are of central importance to the mission of the LarKC project are *robustness* (both with respect to failures of components and incomplete, uncertain or noisy data) and *scalability* (the ability to provide a given functionality on different and potentially very large amounts of data). Thus, any effort to evaluate the project will need to take these aspects into consideration.

Finally, the explication of the project goals need to address the issue of LarKC's *audience*. Ultimately, the usefulness of any practical project depends on it being used productively by a community of users. If LarKC is to succeed it is because multiple social groups are contributing to it and will begin using the platform for tasks that include or transcend the ones currently envisaged. Beyond the interests of the project sponsors, we can distinguish between the following stakeholders:

- LarKC researchers: The various teams of developers involved in the LarKC project;
- Early adopters: Researchers not funded by the LarKC-grant, but using the platform or some of its components for related projects.
- LarKC users: People conducting searches in an area of one of the use cases.
- Long-term adopters are researchers or businesses that transfer LarKC or some of its components to other domains or develop business models on the basis of LarKC-based technology.

Of particular interest for evaluation purposes are the experiences of LarKC users and long-term adopters. Rather than being interested in the platform development itself these user groups will primarily care about the usability of LarKC and the utility of the obtained results for their own purposes. In addition to the sheer number of users and adopters their satisfaction and willingness to invest into using LarKC will be a valuable indicator of LarKC's degree of success.

Metrics

Based on the general and specific goals identified in the previous section we can identify a first set of quantitative dimensions that need to be measured in order to evaluate LarKC. Such dimensions are:

1. *Speed* refers to how quickly is some particular step or query is completed. Although measuring duration seems relatively simple, speed partly depends on the nature of the query, the amount of data considered and the computing resources employed. Thus, to make measurements of speed comparable they need to be expressed in relative terms, e.g., by computing the ratio of time to data (RDF triples) or time, data and computational resources (number of cluster nodes used to compute a solution to a query).

2. *Accuracy* concerns the quality of a result. Importantly, this measure depends on the type of component investigated. For instance, in the context of selection and retrieval plug-ins the typical accuracy measures are precision and recall (see the following chapter). By contrast, traditional measures of accuracy in the context of reasoning modules are soundness and completeness (see the subsequent chapter). In addition, the specific requirements of LarKC to deal with large amounts of potentially inconsistent and incomplete data sets will require non-standard extensions to these traditional measures. Novel reasoning techniques like approximate reasoning and anytime performance proles (see also Deliverable 4.7.1 for initial ideas on measuring and evaluating these techniques).

3. *Robustness* is the ability of a process to yield meaningful results despite potential breakdowns of components and uncertain, incomplete or noisy data. A measurement of robustness can be conducted in several ways:

   (a) by intentionally lesioning the system (by disabling parts of or removing entire components, e.g., particular selector or reasoning plug-ins);

   (b) by degrading the data repositories or data integrity on which a process operates;

   (c) by constraining the temporal or computational resources of a particular computation.

4. *Scalability* refers to the platform's ability to provide a given functionality for different and potentially very large amounts of data. Any measurement of scalability will have to vary the size of the available data (e.g., in millions of RDF triples) and assess changes in speed, accuracy, robustness or quality as a function of this change.

5. *Usability* and *user satisfaction* are measures that address the needs and goals of LarKC adopters and users. Initially, usability can be operationalized as the number of people that use and adopt the platform to their purposes. As the number of users and adopters grows it will be instructive to query these communities about their levels of satisfaction and suggestions for further improvements.

6. *Quality* and *impact* are overall measures of the "goodness" of a thing, process or project—its degree of success and excellence. For practical purposes, quality can be defined in terms of the other measures, i.e., as having high values on the dimensions of speed, accuracy, robustness and scalability, as well as having a large community of satisfied adopters and users. Scientific

impact can be measured by the number of project-related publications and citations.

Just as accuracy can refer to different aspects depending on the particular object whose accuracy is to be assessed we need to distinguish between different *levels of analysis* for the other metrics as well. Specifically, we could measure the quantities just described on the following levels:

1. the overall platform, including many different data repositories, workflows, plug-ins and user queries;
2. a particular use case, involving multiple workflows and plug-ins;
3. a particular workflow or configuration;
4. a particular plug-in;
5. a particular solution to a particular query.

Determining the appropriate level of analysis will depend on the details of a particular question. If the platform instrumentation allows measurement and assessment on the most fine-grained level it will be possible to abstract to the more general levels (but see some complications described in Section 2.2.2 below).

Assessment of Value

An overall assessment of value of the LarKC project is no longer part of the formative evaluation by its partners and contributors but will be achieved through a summative evaluation by its reviewers and sponsor. Beyond the project's scientific impact its practical value can be quantified by the number of adopters and active users. Ultimately, the usefulness of the project will be determined by its uses and users.

Using User Feedback Rather than just providing final judgments on the overall merit of a project user feedback can be used productively during the development process. A convenient way of collecting evaluative user feedback is to automatically prompt users to submit quality ratings upon the completion of a query. For instance, simple Likert scales (that ask for a numeric value between 0 and 10 or prompt the user to point to a value on a linear scale) could be provided for certain key dimensions, like response speed, quality of the obtained solution, and overall satisfaction with the system. As the number of data points obtained would grow with a growing user community it may be possible and—to minimize user annoyance—advisable to sample only a sub-set of all users and prompt them only with a sub-set of dimensions. However, if the content of the original query, information about the system setup that produced a given response and these ratings were logged in a global database such user feedback could provide valuable pointers towards the strengths and weaknesses of platform components. A type of experiment could be to swap in and out particular components and observe the effects of this manipulation of the user feedback.

Another way of using human judgments to improve the LarKC system is to employ several alternative workflows in parallel and let users choose between the results obtained. Side-by-side comparisons of this kind are sometimes implemented to compare the results of different search engines,[1] but they could also be used to compare the results of different system configurations, such as different plug-ins or data repositories.

Openness for Unexpected Uses It is often said that an assessment of quality or value is as much of an art as a science and requires both experience and wisdom. Given the scope of LarKC, it is not just possible but likely that not all practical consequences and uses of LarKC were foreseen and foreseeable from the start. Any evaluative effort needs to be open to the fact that some of LarKC's uses are currently unknown. We need to keep an open mind and adapt our ongoing efforts of evaluation to new objectives as the project unfolds.

2.2.2 Heuristic Problem Solving in LarKC

Just as the performance of the LarKC platform needs to be measured on multiple dimensions and levels the notion of heuristics is embedded in LarKC in multiple ways.

Heuristic Plug-Ins

On the level of individual plug-ins developers use and implement heuristics in the sense of strategies that are computationally efficient and yield satisfactory results. Granular reasoning (as described in LarKC Deliverable 2.3.1 and Zhong et al., 2008) can be considered a heuristic method to facilitate reasoning by zooming into a subject matter on different levels of resolution. Likewise, techniques for approximate reasoning and anytime reasoning (as described in this document and Deliverable 4.7.1) constitute heuristic approaches that dispense with some of the requirements of classical logic in the service for computational or practical purposes. One way to evaluate these techniques is by comparing their results with alternative results obtained from different reasoners. A difficulty of such comparisons is that different methods are likely to differ on multiple dimensions. For instance, some Reasoner A may yield a more complete set of statements than an alternative Reasoner B but may simultaneously require more time or be more volatile with respect to inconsistent or noisy data. Unless there is an explicit norm how the different dimensions of completeness, speed and robustness are to be integrated it is unclear which reasoner is superior. As the solution to such trade-offs also depend on the sources of data, the selection and transformation of data prior to the reasoner and a particular use case a cautious approach is to not attempt an integration, but measure all dimensions separately.

Heuristic Deciders

A second sense in which LarKC will implement aspects of heuristic problem solving is in the rules that determine the duration for which processes continue to run. Although stopping rules were previously mentioned as a key component of

[1]See e.g., `http://blindsearch.fejus.com` and `http://www.blackdog.ie/google-bing/` for a comparison between alternative search engines (validity of URLs verified on 2009-09-08).

a well-defined heuristic (see page 9 above) they also constitute genuine heuristics in themselves. As the time and complexity of many computations increases with the size of the data set to-be-searched rules that decide when to stop searching or when to interrupt one task in favor of another are increasingly important. If a plot of performance increases as a function of time on some critical dimension shows the characteristics of diminishing returns it may be possible to increase the overall efficiency of the system by implementing heuristic stopping rules (see Deliverables 4.2.1 and 4.2.2 for examples of such rules). As before, a critical evaluation of the success of such rules requires that their benefits (e.g., in computational efficiency) are traded-off against their costs (e.g., loss of some data). As the weights of such a trade-off cannot be specified *a priori* and independently of a specific problem our current recommendation is to measure all available aspects of the problem and determine their integration when the need arises in a particular context.

Heuristic Workflows

A third way in which the LarKC platform employs a form of heuristic problem solving is the way in which multiple components are assembled in order to respond to a user query. Regardless whether a particular workflow was hand-coded by a developer or designed on-the-fly by an automated LarKC decider the workflow itself can be considered a heuristic strategy with the goal to discover a solution (or solutions) to the given problem.

The very notion of a plug-in makes it apparent that a choice between multiple alternative components immediately poses a selection and credit assignment problem. For instance, if a particular problem could be addressed by n alternative selectors and m alternative reasoners there exist $n \times m$ alternative workflows that involve exactly one of each plug-in types. To assess the performance characteristics of each of these workflows is no trivial matter, especially since it is possible that the effects of sequential plug-ins are not merely additive (e.g., the total execution time is equivalent to the sum of all component times) but may be subject to manifold interactions (e.g., when a particular Reasoner works well with the output of a specific Selector A, but poorly with the output of another Selector B). As such interactions cannot always be foreseen when designing Selectors A and B the prudent approach with regards to evaluation is to instrument all components and measure and compare different workflows once the components are in place and used repeatedly to solve practical problems.

2.2.3 Evaluation of Heuristic Problem Solving

As heuristics are methods or procedures to solve problems the evaluation of LarKC's heuristic components can be based on the same same principles, metrics and methods that have already been outlined above (see Section 2.2.1, pages 9ff.). Nevertheless, we will highlight three methodologies that are of particular importance for the evaluation of heuristic problem solving:

1. *Comparison with alternative methods*: For heuristics that provide computational short-cuts to arrive at satisfactory results at low costs (in terms of the time or computing resources invested) it is always instructive to quantify

both the benefits and costs of the heuristic compared to alternative methods, e.g., an established optimization method, multiple (sizes of) data sets, different system configurations, or the results of alternative strategies.

2. *User feedback*: To assess the degree to which different values on multiple parameters (like speed, recall, precision etc.) matter to real-world applications it will be important to incorporate evaluative user feedback (as outlined in Section 2.2.1, page 14).

3. *Cross-validation*: If a large number of problems with expert solutions exist for a particular use case the methodology of cross-validation would facilitate both platform design and evaluation. In an initial design phase, developers would receive a random sub-set of all problems to design and fine-tune a LarKC workflow that yields results that resemble the expert solutions as closely as possible. In a second test phase, the resulting workflow could be evaluated on its responses to the remaining problems that were not available to the developer during the design phase. This cyclical process of test-adjust-test can be subdivided into smaller steps and implicitly reflects the standard approach of most developers. But in project of the scale of LarKC—in which the responsibilities about use case content and platform design are distributed across multiple people and research groups—it may be useful to explicitly install a collaborative cross-validation effort.

2.3 Summary

In this chapter we clarified the concepts of *measurement*, *evaluation* and *heuristic problem solving* and explicated their meaning in the context of the LarKC project.

Any large-scale evaluative effort serves multiple functions and the interests of many stakeholders. Our primary interest in the context of this deliverable is a formative evaluation framework that is designed by the members of the LarKC project and aims to identify project strengths and weaknesses, focus managerial and research efforts, and measure progress towards achieving the project goals. Such a framework can also serve as the foundation for a subsequent summative evaluation.

Key components of an evaluation are the identification of objectives, metrics, and the attribution of value to the measurements obtained. With respect to LarKC we outline important project goals and key dimensions on which the platform's performance can be quantified. With respect to evaluative judgments we argue that the negotiation of trade-offs between multiple dimensions should be addressed at a later point, when they can be informed by a larger user base. At the current stage of the project the main focus should be on instrumenting the components of the platform and collecting the associated measurements in a systematic way. As the usefulness of LarKC will be determined by its users and uses we emphasize the importance of incorporating user feedback throughout the development and evaluation of the project.

The notion of heuristic problem solving is used in many different contexts and meanings. We identify several usages and outline three ways in which the concept can be applied to LarKC: as methods used in plug-in design, as stopping rules implemented in deciders, and as workflows that are configured to solve particular problems. The evaluation of heuristic problem solving follows the same methods and principles as evaluation in general.

3 Evaluation of Identify and Select Plug-ins

This chapter includes a specification of different criteria and techniques for evaluating selection and retrieval plugins. The role of retrieval and selection in LarKC is to dynamically reduce or expand the data set we are working with depending on factors such as cost of processing or confidence in result. In other words, the task is to return all those triples from a repository which are the most relevant to the context (e.g., reasoning history). To do this we exploit methods from Information Retrieval (IR), machine learning, cognitive science, and stream management systems.

Although there is a similarity between the selection task in LarKC and classic IR, it is important to note that unlike IR which deals with documents, selection tasks deal with structured data only, i.e. RDF triples. Therefore, we cannot use classical IR evaluation measures, without adapting them. Moreover, evaluation of selection and retrieval plugins in LarKC is a complex process as it requires addressing several challenges (detailed discussion available in *D2.1.2 Selection: Experimental Apparatus, Chapter 5*):

- The LarKC platform will be used for different scenarios and will be applied to various usecases. Therefore evaluation needs to be complete in this sense.
- It is very hard to measure the success of selection plugins, as it can only be measured with respect to a task that has obvious relevance for humans. However, it is not feasible for humans to transfer the original KB and the subset into their minds in order to judge the relevance of the subset. In this sense, evaluating subsetting in isolation becomes unreilable. However, evaluating more concrete tasks which depend on subsetting sounds feasible.
- Further emphasizing the problem described in the previous point, RDF knowledge bases are not written in a language understandable for humans. It is very hard for general population or domain experts from the specific field to understand and evaluate the structure represented as RDF triples.
- To the best of our knowledge, established 3rd party evaluation programmes do not exist for the task such as selection in LarKC.

In the rest of this chapter, we discuss evaluation measures such as speed and accuracy, followed by datasets, and a more detailed discussion on the evaluation of selection plugins within the project usecases.

3.1 Speed

Measuring speed in the context of the selection plugins means measuring number of statements per second. This measure cannot be observed in isolation, and needs to

- be expressed relative to computational resource usage,
- consider the relevance of the output subset expressed through accuracy.

With regards to the types of statements which are covered by this measure, we consider:

- number of statements per second considered for inclusion in the output subset, or/and
- number of statements per second which have been removed.

3.2 Accuracy

As selection and ranking problems bear a number of similarities to that of Information Retrieval (IR) we will exploit a number of evaluation methods from that field, namely precision and recall.

Precision measures the number of correctly identified items as a percentage of the number of items identified. In other words, it measures how many of the items that the system identified were actually correct, regardless of whether it also failed to retrieve correct items. The higher the Precision, the better the system is at ensuring that what has been identified is correct.

Recall measures the number of correctly identified items as a percentage of the total number of correct items. In other words, it measures how many of the items that should have been identified actually were identified, regardless of how many spurious identifications were made. The higher the Recall rate, the better the system is at not missing correct items.

When measuring accuracy of the selection task, recall is more important than precision. The reason is the subsetting task being the first step which is performed before other processes. If the subsetting task removes relevant triples, the following processes will be less accurate and may provide only approximate solutions which in turn affects the quality of answers computed by the workflow.

However, traditional IR measures such as precision and recall need to be adapted in order to suit the selection and retrieval task in LarKC, which deals with RDF triples, not documents. More precisely, in order to calculate these measures, we need to know the size of the dataset in advance, and the exact set that solves the query. Even if we perform these types of measurements on the small test sets, this would require manual checking the relevance of RDF triples. As we have discussed before, humans might not be able to correctly assign the relevance to the RDF triples as although the triple could seem irrelevant for a human, it might be crucial for the successful operation of a subsequent plugin such as a reasoner. Moreover, the KB can change over time, which means that the relevant subset might change as well. All these reasons make the expectations from humans to judge the relevance of a huge RDF subset unrealistic.

Therefore, as we need to involve humans and design human-understandable tasks in order to evaluate the subsetting in LarKC, we will conduct *extrinsic* task-based evaluation based on the usecases so that we can express the relevance on a more abstract level, without engaging users in the process of understanding and judging the relevance of the RDF subset. An *extrinsic* evaluation evaluates the component as part of a complete system, in its actual use case setting. This is discussed later in Section 3.5.

We will also consider *intrinsic* evaluation. An intrinsic evaluation is designed to test a single component in isolation, holding all external factors constant, and independently of its actual use. Details are discussed later in Section 3.4

As the use cases are major providers of test data for the LarKC platform, we will first describe the datasets with which the evaluation will be performed.

3.3 Datasets

The life sciences are a major producer of data, information, and knowledge at terabyte scales and above. Witness the vast quantities of raw genome data produced by the many species specific genome projects, the ontologies built to help describe this data (some of the largest ontologies built to date), and the burgeoning gene- and prote- omics literature. The datasets that we use in LarKC therefore provide good vehicles on which to test notions of "web-scale" and "ceiling free" reasoning.

We will make use of two major data sources when performing cancer research use case evaluations:

- A large triple store (Linked Life Data - LLD), as described in D7b.1.1a "Requirements summary and data repository", Section 6.2. LLD will be a primary, single resource in the intrinsic evaluation of selection and retrieval plugins. For extrinsic evaluation, it will be used in combination with other data sources.

- A corpus of annotated biomedical literature, as described in D7b.1.1a "Requirements summary and data repository", Section 6.3.1 and Section 6.3.2. We will annotate the full 20 million abstracts of PubMED/MEDLINE [1]. When represented as triples in LLD, this will provide further material, particularly for the extrinsic evaluations discussed below.

In addition, the use case is a source of end-user queries, that may be of use in both extrinsic evaluations, and in providing queries for intrinsic evaluation of selection and retrieval plugins. Example queries are described in D7b.1.1a "Requirements summary and data repository", Section 5.3.4.

3.4 Intrinsic evaluation

An intrinsic evaluation is designed to test a single component in isolation, holding all external factors constant. The evaluation is independent of actual use, and does not involve any objective measure of usefulness, or any subjective measure of end user satisfaction. This gives a measure of performance with respect to the components functionality, independent of utility. For example, a selection plugin may be evaluated against a synthetic dataset, with synthetic queries.

For intrinsic evaluation, we will concentrate on the most straightforward derivations of IR precision and recall metrics, together with computational efficiency. These metrics will be measured with respect to triple selection across evaluation data sets. Wherever possible, these datasets will be drawn from the use cases. Datasets may, however, need to be specific to a particular plugin.

[1] `http://www.ncbi.nlm.nih.gov/pubmed/`

Each plugin will be evaluated within a fixed configuration of the LarKC system. For each plugin evaluation, we will use a comparison against a baseline to show relative changes in performance for each parameter. For this purpose, the baseline plugins defined in Deliverable D2.2.1, 2.5.1 "Month 12 Selection Components (report accompanying two software deliverables)" will be used. Evaluation will proceed as follows:

1. Define a fixed configuration of the LarKC platform.
2. For each evaluation dataset and for each *baseline* plugin
    (a) Run evaluation queries over *baseline* plugins
    (b) Log evaluation metrics
    (c) For each *evaluation* plugin
        i. Run evaluation queries over evaluation plugin
        ii. Log evaluation metrics
        iii. Compute change in metrics

3.5 Extrinsic evaluation

An extrinsic evaluation evaluates the component as part of a complete system, in its final use setting. Tipically, this would be conducted as a task-based evaluation with subjects, who will be asked to perform the specific tasks using the system. The measures we can derive from such an evaluation are:

- *effectiveness* – the ability of users to complete tasks using the system, and the quality of output of these tasks i.e. was the same task faster, cheaper, or more accurate?
- *efficiency* – the level of resource consumed in performing tasks i.e. the time the subjects spent on the task, and
- *satisfaction* – the user's subjective reaction to using the system

We will consider two use case scenarios from the cancer research use case, detailed below.

3.5.1 Evaluation in the Genome Wide Association Study scenario

The Genome Wide Association Study (GWAS) scenario is desccribed fully in D7b.1.1a "Requirements summary and data repository", Chapter 3.

In GWAS, samples are tested from thousands of subjects with the disease in question, and thousands of disease-free controls. Each sample is tested with many hundreds of thousands of gene markers. If a marker is found more frequently in disease samples as opposed to control samples, then perhaps genes close to that marker are associated with the disease. Unfortunately, the statistical power of the technique is limited. This is overcome by using larger and larger numbers of experimental samples and controls. The aim of LarKC in GWAS is to find the prior probability that a given gene is associated with a disease, and to improve the statistical power of GWAS data analysis on small numbers of samples, through using this prior in a Bayesian model. The prior is calculated using:

- A set of keywords selected by domain experts to represent the disease
- All research literature abstracts in MEDLINE, semantically annotated
- A selection of biomedical knowledge sources in LLD, as described above
- A set of queries relating the keywords, literature and knoweldge sources

The use case may be evaluated by virtue of the fact that we have an existing non-LarKC method, and existing experimental data. We aim to use LarKC on subsets of the existing experimental data, and determine if this subset could be used to produce the same results as the full set using the existing non-LarKC method. Our measure will be the percentage of data required for LarKC to produce the same results as the non-LarKC method. We will use multiple (100) random selections of datasets.

3.5.2 Evaluation in the Monograph scenario

The Monograph scenario is described fully in D7b.1.1a "Requirements summary and data repository", Chapter 2.

The IARC monographs are a series of volumes evaluating potential carcinogens. They are definitive, and encyclopaedic. Each monograph is based on a review of: all published human and animal studies; representative studies of the molecular basis of carcinogenesis. Monograph scientists identify: all relevant papers for review; relevant international experts to carry out the review and write the monograph. Monograph scientists currently use typical IR methods to search for the relevant literature. The aim of LarKC is to improve retrieval by the use of semantic annotation and network analysis techniques.

Evaluation of this scenario will make use of a task-based as described in the deliverable "D7b.1.1b Iteration evaluation methodology and report template". The methodology will make use of a Likert System Usability Scale, along which users will be asked to score performance and ease of obtaining results for a LarKC assisted literature search, and for a non-larKC assisted literature search.

3.6 Workflow-based evaluation

Evaluation of the selection task within LarKC platform will enable us to get an objective view of the task quality by comparing:

- the workflow *speed/resource consumption* with and without selection, and also
- the *accuracy* of the result of the workflow with and without selection

4 Evaluation of Transformation Plug-ins

In this chapter we explore some evaluation criteria and measures that can be utilized to evaluate the transformation techniques and plug-ins developed in LarKC. At first we give an overview of the transformation techniques and plug-ins. Then we briefly review related machine learning algorithms and show the workflow of the approach applied for the transformation plug-ins. Afterwards, we provide an insight into the evaluation in terms of efficiency, quality, robustness and scalability and discuss some state-of-the-art evaluation methods in areas of machine learning and information retrieval. Since there is no established executable transformation plug-in yet, we focus on the evaluation criterion and measures which we used by now in our experimental tests. Finally we conclude the chapter. Note that WP3 “Abstraction and Learning” consists of two parts: Machine Learning and Data Streaming and that here we solely concern the former whereas the latter will be described in another separated chapter.

4.1 Overview

The objective of WP3 is to contribute scalable solutions to abstraction and learning to LarKC. Abstraction concerns the derivation of suitable data representations for reasoning and learning, while for learning we will extend existing machine learning algorithms to be applicable in the LarKC framework and explore suitable reasoning tasks in context of the LarKC use cases.

The techniques and plug-ins developed in WP3 can be used to transfer convenient semantic knowledge base (KB) into probabilistic semantic KB, where RDF statements contain certain probability how likely they might exist or be true. This transformation could be considered as inductive materialization by using machine learning. Traditionally, machine learning requires a domain expert for the feature definition, for the selection and the application of learning algorithms and for the interpretation of the results. The eventual goal of plug-ins of WP3 is a machine learning approach that is appropriate to the data situation on SW but that is as easy to use as logical reasoning. We summarize the challenges of the transformation techniques and plug-ins in LarKC as follows:

- They should scale well with the size of the SW.
- They should deal with sparse data and missing information.
- The statements and their probabilities, which are predicted by machine learning, should easily be integrated into SPARQL-type querying.
- They should be “push-button” requiring a minimum of user intervention.

4.2 The Transformation Approach

A number of algorithms have been successfully proposed in the past decade for learning in Semantic Web (SW), many of which are based on recent work in statistical relational learning. Two families of approaches are special worth to mention:

global probabilistic models (e.g. Markov Logic Networks) and conditional models (e.g. Inductive Logic Programming). Each of them has strength and weakness. We pursue a compromise between two families of models. As in some of the global probabilistic models, we introduce probabilistic nodes whose states reflect the truth value of the corresponding statements. We derive a data matrix for model training. Such a data matrix is typically high-dimensional and sparse. We apply matrix completion techniques for estimating the missing information.

How does this learning approach work? Given a semantic KB, user defines the statistical unit and the population. Based on the definition RDF triples are transformed into data matrix with two dimensions: Rows are statistical units (key entities), while columns represent features of the data. This RDF-to-matrix transformation can be considered, from machine learning point of view, as a preprocessing step. We perform sub sampling and feature collection by using SPARQL-like queries and we achieve feature reduction afterwards. With the data matrix we train models and apply the models to estimate statements for the unknown entries in the sampled data matrix (transduction) and more importantly to estimate unknown statements in the whole population (induction). The estimated probabilistic statements can be stored in the KB as weighted triples, so that they can be retrieved together with existing statements via SPARQL-like queries. The approach is proposed in (Tresp, Huang, Bundschus, & Rettinger, 2009).

4.3 Efficiency

So far we have got a notion of the learning approach. The most time consuming step in the workflow is the model training. However, the training is usually processed offline (not in runtime). The estimation of the probability of unknown but potentially true statements can be done online or offline. It depends on concrete application scenarios. In case of inductive materialization, for example, we perform both training and prediction offline, so that the efficiency of querying is not debased. Since the approach is based on matrix completion techniques, predicting the truth value of a certain statement takes often time in $O(n)$ complexity order where n is the dimensionality of the features. Some probabilistic generative models might need longer time to do prediction and it is dependent on various factors: again dimension of features, program language, implementation of models and of course the intrinsic characters of models. In any case, we can evaluate the efficiency of the transformation plug-ins by using any standard criterium. For instance, how many statements are estimated per second.

4.4 Quality

In WP3 we take open world assumption which means that no existing statements are unknown but not false. Therefore the task here is to estimate the probability of potential statements by exploring regularities in the semantic data using machine learning. In order to evaluate the performance of learnt models we split data into training set and test set by randomly selecting some known statements for test and setting the corresponding matrix entries to zero, to be treated as unknown. In the test phase we then predict all unknown statements, including the test statements.

The test statements should obtain a higher likelihood value, if compared to the other unknown entries. Sorting statements by descending estimated probability we get a rank list in that the test statements should appear in top positions.

In our preliminary experiments we use the normalized discounted cumulative gain (NDCG) (Jarvelin & Kekalainen, 2000) to evaluate the quality of rank list predicted by learning models. NDCG is calculated by summing over all the gains along the rank list R with a log discount factor as $\mathrm{NDCG}(R) = Z\sum_k(2^{r(k)} - 1/\log(1+k))$, where $r(k)$ denotes the target label for the k-th ranked item in R, and Z is chosen such that a perfect ranking obtains value 1. To focus more on the top-ranked items, we also consider the *NDCG@n* which only counts the top n items in the rank list. These scores are averaged over all functions for comparison. The better an algorithm, the higher would the test statement be ranked.

In general, there is a number of methods in areas of machine learning and information retrieval to evaluate the quality of predictive models. However, as we assume that unobserved statements are unknown, there are no negative examples (false) at all and many evaluation methods are not appropriate in this case. For example, precision and recall mentioned in Chapter 3, as well as accuracy and F_1 measure. The latter is a combination of precision and recall, formulated as $F_1 = 2*precision*recall/(precision+recall)$, because precision and recall are negative correlated. Another family of evaluation measures are based on the residual error. They are suitable in case of numeric truth values of statements (e.g. movie ratings from one to five stars) instead of binary values. An example in this family is the rooted mean squared error (RMSE) which is a good measure of the differences between estimations and true values. The RMSE is calculated by drawing the square root of the average squared error on each residual.

4.5 Scalability

The size of the training data set is under the control of the user by means of defining the population and the statistical unit. Thereby the data matrix is typically independent or only weakly dependent on the overall size of given semantic KB and in consequence the time consume and feasibility of model training is essentially independent of the overall size of the KB. The learning approach presented in Section 4.2 achieves scalability via two means. First, sub sampling techniques lead to scalable solutions. Second, scalable algorithms can deal with large data matrix and are only sensitive to the number of non-zero entries. Theoretically we can say that the transformation plug-ins will be scalable to arbitrary size of data. If the size of the population becomes really huge and the characteristics of the test set becomes very different from the characteristics of the training set prediction becomes harder: e.g., random people in the world do not have friends in common.

4.6 Robustness

To evaluate the robustness of a certain machine learning model one normally tests the model based on more than one data sets from different domains. Taken one data set, one can use cross validation repeating data split (training data set and test data set: disjunct subsets of that data set), so that one averages experimental results on different subsets and evaluates the performance of the model with the mean value and corresponding confidence interval. In addition, due to the application of sub sampling techniques we repeated sampling process in our experiments for several times.

Another aspect of the robustness (see Chapter 2) concerns the capacity of handling with incomplete, uncertain or noisy data. It is exactly the nature of the matrix completion techniques.

4.7 Summary

In this chapter we explored the key idea and the workflow of the machine learning approach applied in transformation plug-ins, which is well suitable for typical SW data situation: large scale, uncertain and highly sparse with missing information. We analyzed characters of the approach with respect to efficiency, quality, scalability and robustness and based on that we presented corresponding evaluation measures. In addition, we briefly discussed in Section 4.4 some related evaluation measures in terms of quality. Concerning usability we would point out that the transformation is to a large degree autonomous and only the statistical unit and the population need to be defined by a user, although user also has possibility to design configuration by her/himself.

5 Evaluation of Reason Plug-ins

In this chapter, we lay the foundations for a statistical approach to evaluating approximate and anytime reasoning algorithms. We will do this in a very abstract manner, which can be made concrete in different ways, depending on the considered use case. At the same time, we will use this statistical perspective to precisely define approximate reasoning notions which to date have remained quite vague. We furthermore show that our mathematical modelling can be used for guiding the development of composed approximate reasoning systems. In the end, our mathematical modelling can be used for rigorous comparative statistical evaluation of approximate reasoning algorithms.

5.1 Introduction

In different application areas of Semantic Technologies, the requirements for reasoning services may be quite distinct; while in certain fields (as in safety-critical technical descriptions) soundness and completeness are to be rated as crucial constraints, in other fields less precise answers could be acceptable if this would result in a faster response behaviour.

In many cases, however, a user will not be willing to wait arbitrarily long for an answer. More likely, she would be prone to accept “controlled inaccuracies” as a tradeoff for quicker response behaviour. However, the current standard reasoning tools (though highly optimized for accurate, i.e., sound and complete reasoning) do not comply with this kind of approach: in an all-or-nothing manner, they provide the whole answer to the problem after the complete computation. It would be desirable, however, to have reasoning systems which can generate good approximate answers in less time, or even provide “anytime behaviour”, which means the capability of yielding approximate answers to reasoning queries during ongoing computation: as time proceeds, the answer will be continuously refined to a more and more accurate state until finally the precise result is reached. Clearly, one has to define this kind of behaviour (and especially the notion of the intermediate inaccuracy) more formally.

It is important to realize that computation time is just one of the resources that an algorithm uses, and that the user might want to economise on in exchange for a reduced quality of answers. In other settings, one might want to reduce other resources such as the amount of memory used by an algorithm (e.g. on memory-limited mobile devices), the amount of user interactions needed to complete the task (in order to put less burden on the user), the amount of data access needed by the algorithm (e.g. in pay-for-access environments), etc. The formal framework presented in this chapter is general enough to cover approximation as a trade-off of quality against *any* resource, time being just a particular (and frequently ocurring) example.

Approximations can take multiple forms. We will now provide some examples and motivatoins for a number of different forms of approximations.

- *Approximating the set of answers: incompleteness* if not all answers are needed to complete a task (e.g. instead of returning all the phone numbers of a taxi company return only one them, i.e. return a single answer instead of multiple answers to a query).

Or, weaker, if a partial set of answers already allows partial completion of the task. For example consider the scenario where an user is interested in comparing the prices of a given item across various internet shops. The more shops the better, but even a partial set of shops allows you to make a relatively good comparison and decision.

- *Approximating the set of answers: unsoundness* when wanting to rapidly exclude a suspected answer: cheaply compute an unsound answer set (= too large); if the suspected answer is not in the computed (too large) answer set, it is certainly not an answer, and can be excluded. This happens when wanting to ensure that no solutions are missed: cheaply compute an unsound answer set (= too large); all correct answers are guaranteed to be included (at the price of having a few extra incorrect answers).

  For example, in systems where a rapid diagnosis for high risk faults is required a valid option is to compute an initial unsound (too large) diagnosis. If this set contains a high risk fault then it is worth spending more resources to see if the same high risk fault is included in a more precise diagnosis. As another example consider a query that ask for the current location in the context of location-aware services. This query is an example of the type of queries users may formulate in the Urban Computing use case. A possible solution would be to compute and return an approximate location rather than the exact location. In this case, an unsound but complete answer set would include multiple locations, one of which is the correct one.

- *Approximating individual answers* When the approximation consists not of reducing a set of correct answers to a smaller set, but when an individual answer is being approximated. For example, in determining the personal profile-category of a customer a more precise profile is better, but even a high level abstract profile-category is better than nothing. Another example is finding a product that satisfies as many requirements of a customer as possible. A product that only satisfies some of the requirements is already an approximate solution.

- *Disjointness of approximate and perfect answers:* Do the approximate answers come from the same domain as the perfect answers or not? In the customer-profile problem, no approximate abstract-class answer would ever qualify as the perfect answer, since we always want a concrete (leaf) class as an answer (hence the domain of approximate solutions and of perfect solutions are disjoint), while in the product-selection problem, the approximate answer to one problem could be the perfect answer to another problem, hence the domain of approximate solutions and of perfect solutions coincide.

- *Two-sided approximation:* When both an upper bound and a lower bound are given, which approximate the perfect answer from two directions (e.g. both complete-unsound and sound-incomplete sets of answers)

5.1.1 Application to Semantic Web

Introducing approximate reasoning in the Semantic Web field is motivated by the following observation: most current specification languages for ontologies are quite expressive, reasoning tasks are supposed to be very costly with respect to time and other resources – this being a crucial problem in the presence of large-scale data. As a prominent example, note that reasoning in most description logics which include general concept inclusion axioms (which is simply standard today, and e.g. the case in OWL DL) is at least EXPTIME complete, and if nominals are involved (as for OWL DL) even NEXPTIME complete. Although those worst case time complexities are not likely to be thoroughly relevant for the average behaviour on real-life problems, this indicates that not every specifiable problem can be solved with moderate effort.

These ideas of approximate reasoning are currently cause of controversial discussions. On the one hand, it is argued that soundness and completeness of Semantic Web reasoning is not to be sacrificed at all, in order to stay within the precise bounds of the specified formal semantics. On the other hand, it is argued that the nature of many emerging Semantic Web applications involves data which is not necessarily entirely accurate, and at the same time is critical in terms of response time, so that sacrificing reasoning precision appears natural (Fensel & Harmelen, 2007).

Another suggestion to avoid the necessity is to restrict knowledge representation to so-called tractable fragments that allow for fast, sound and complete reasoning. Although this might be useful in scenarios where all essential knowledge can be modelled within the restricted fragment, in general there are strong arguments in favor of the usage of expressive formalisms:

- Real and comprehensive declarative modelling should be possible. A content expert wanting to describe a domain as comprehensively and as precisely as possible will not want to worry about limiting scalability or computability effects.

- As research proceeds, more efficient reasoning algorithms might become available that could be able to more efficiently deal with expressive specification formalisms. Having elaborated specifications at hand enables reuse of the knowledge in a more advanced way.

- Finally, elaborated knowledge specifications using expressive logics can reduce engineering effort by horizontal reuse: Knowledge bases could then be employed for different purposes because the knowledge is already there. However, if only shallow modelling is used, updates would require overhead effort.

5.1.2 Contributions of this Chapter

From our perspective, it depends on the specifics of the problem at hand whether approximate reasoning solutions can or should be used. We see clear potential in the fields of information retrieval, semantic search, as well as ontology engineering support, to name just a few examples.

At the same time, however, we would like to advocate that allowing for unsound and/or incomplete reasoning procedures in such applications must not lead to arbitrary "guessing" or to deduction algorithms which are not well-understood. Quite on the contrary, we argue that in particular for approximate reasoning, it is of utmost importance to provide ways of determining how feasible the approximations are, i.e. of what quality the answers given by such algorithms can be expected to be.

Obviously, soundness and completeness with respect to the given formal semantics of the underlying knowledge representation languages cannot be used as a measure for assessing the quality of approximate reasoning procedures. Instead, they must be evaluated experimentally, and analysed by statistical means.

The field of knowledge representation has a long history of studying approximate reasoning methods for propositional and first-order logic (see e.g. (Dowling & Gallier, 1984; Horvitz, 1987; Selman & Kautz, 1991; Schaerf & Cadoli, 1995; Cadoli & Schaerf, 1995; Dalal, 1998; Cadoli & Scarcello, 2000; Harmelen & Teije, 2000; Groot, Teije, & Harmelen, 2004)). These are only now being applied in the context of OWL reasoning for Semantic Web technologies. Notable recent papers in this area are (Stuckenschmidt & Harmelen, 2002; Horrocks, Li, Turi, & Bechhofer, 2004; Groot, Stuckenschmidt, & Wache, 2005; Perry Groot, 2005; Hitzler & Vrandecic, 2005; Pan & Thomas, 2007; Holger Wache, 2005; Stuckenschmidt, 2007). The methods in these papers are very diverse, and no overall framework exists for formally describing and comparing the quality of these approaches to approximate reasoning.

In this chapter, we lay the foundations for a statistical approach to evaluating approximate and anytime reasoning algorithms. We will do this in a very abstract manner, which can be made concrete in different ways, depending on the considered use case. At the same time, we will use this statistical perspective to precisely define approximate reasoning notions which to date have remained quite vague. We furthermore show that our mathematical modelling can be used for guiding the development of composed approximate reasoning systems. In the end, our mathematical modelling can be used for rigorous comparative statistical evaluation of approximate reasoning algorithms.

As a word of caution, let us remark that the notion *approximate reasoning* bears two different meanings in two different communities. Often, the notion is associated with *uncertainty reasoning*, e.g. in the sense of fuzzy or probabilistic approaches. The notion of approximate reasoning we use in this document refers to approximate reasoning algorithms on data which is *not* uncertain in this sense.[1]

This chapter is structured as follows. In Section 5.2, we will establish a mathematical framework as a foundation for approximate reasoning notions and evaluation. In Section 5.4 we will discuss composition of approximate reasoning algorithms from the perspective of our framework. In Section 5.5 we show how to instantiate our framework by means of an example. We conclude in Section 5.6.

[1]Perhaps introducing the notion of *qualitative approximate reasoning* – to replace *approximate reasoning* in our sense – would help to clarify matters. In order to be consistent with the literature, however, we prefer to use the established notion for now.

5.2 A Framework for Approximate Reasoning

In this section, we establish a mathematical framework which allows us to provide a formal basis for central notions of the field and establish guidance for lines of further research in that area. We first define some basic notion (input and output spaces), and then introduce error functions defined over these spaces. Next, we introduce some formal machinery to talk about algorithms. We then formalise the notion of defects of an algorithm. This then allows us to compare the quality of the output of algorithms.

5.2.1 Input and Output Spaces

First, let us stipulate some abbreviations which we will use in the sequel: let $\mathbb{R}^+ = \{x \in \mathbb{R} : x \geq 0\}$ and $\mathbb{R}^+_\infty = \{x \in \mathbb{R} : x \geq 0\} \cup \{+\infty\}$.

First of all, we have to come up with a general and generic formalization of the notion of a *reasoning task*. Intuitively, this is just a *question* (or query) posed to a system that manages a knowledge base, which is supposed to deliver an *answer* after some processing *time*. The (maybe gradual) validity of the given answer can be evaluated by investigating its compliance with an abstract semantics. We will extend this classical conceptualization in the following way: we allow an algorithm to – roughly speaking – change or refine its output as time proceeds, thus capturing the notion of *anytime behaviour*, as a central concept in approximate reasoning. Yet in doing so, we have to take care not to lose the possibility of formalizing "classical" termination. We solve this by stipulating that every output of the system shall be accompanied by the information, whether this output is the ultimate one.

In the sequel we will formalize these intuitions. By the term INPUT SPACE we denote the set of possible concrete reasoning tasks. Formally, we define the input space as a probability space (Ω, P), where Ω is some set (of inputs) and P is a probability measure on Ω. The probability $P(\omega)$ encodes how often a specific input (knowledge base, query) ω occurs in practice, resp. how relevant it is for practical purposes. Naturally, information about the probability distribution of inputs will be difficult to obtain in practice (since, e.g., in general there can be infinitely many different inputs). Thus, heuristics or "rules of thumb", like for instance giving short queries a higher probability than long ones, or using some kind of established benchmarks, will have to be used until more systematic data is available.

The use of having a probability on the set of inputs is quite obvious: as already stated before, correctness of results cannot be guaranteed in the approximate case. So in order to estimate how good an algorithm performs in practice, it is not only important, how much the given answer to a specific input deviates from the correct one, but also how likely (or: how often) that particular input will be given to the system. Certainly, a wrong (or strongly deviant) answer to an input will be more tolerable if the query occurs less often.

For actual evaluations, one will often use a discrete probability space. For the general case – for developing the theory in the sequel – we will assume that all occurring functions are measurable (i.e. integrals over them exist), which is obviously a very mild assumption from a computer science perspective.

The OUTPUT SPACE comprises all possible answers to any of the problems from the input space. In our abstract framework, we define it simply as a set X.

5.2.2 Error functions

A function $e : X \times X \to \mathbb{R}^+$ – which we call *error function* – gives a quantitative measure as to what extent an output deviates from the desired output (as given by a sound and complete algorithm). More precisely, the real number $e(x, y)$ stands for the error in the answer x, assuming that y would be the correct answer. For all $x \in X$ we assume $e(x, x) = 0$, but we place no further constraints on e. It will be determined by the problem under investigation, though a suitable example could be $1 - f$, where f is the *f-measure* as known from information retrieval. In certain cases, it might also be useful to put more constraints on the error function, one could e.g. require it to be a metric,[2] if the output space has a structure where this seems reasonable.

We will assess the usefulness of an approximate reasoning algorithm mainly by looking at two aspects: Runtime and error when computing an answer. By introducing the error function, we are able to formalize the fact that out of two wrong answers one might still be better than the other since it is "closer" to the correct result. While this might not seem to make much sense in some cases (e.g. when considering the output set $\{\mathit{true}, \mathit{false}\}$ or other nominal scales[3]), it might be quite valuable in others: When we consider an instance retrieval task, the outputs will be sets of domain individuals. Obviously, one would be more satisfied with an answer where just one element out of hundred is missing (compared to the correct answer) than with a set containing, say, only non-instances.

We assume X to contain a distinguished element $\bot$ which denotes *no output*. This is an issue of "backward compatibility", since classical algorithms – and also many approximate reasoning algorithms – usually do not display any output until termination. So, to include them into our framework, we define them to deliver $\bot$ before giving the ultimate result. $\bot$ will also be used as output value in case the algorithm does not terminate on the given input.

Since by this definition, $\bot$ contains no real information, one could argue about additional constraints for the error function with respect to this distinguished element, e.g., $e(\bot, y) \geq \sup_{x \in X}\{e(x, y)\}$ or even $e(\bot, y) \geq \sup_{x,z \in X}\{e(x, z)\}$. We do not need to impose these in general, however.

5.2.3 Algorithms

After having formalized inputs and outputs for problems, we now come to the actual algorithms. In order not to unnecessarily overcomplicate our formal considerations, we make some additional assumptions: We assume that hardware etc. is fixed, i.e., in our abstraction, an algorithm is always considered to include the hard- and software environment it is run in. I.e., we can, for example, assign any algorithm-input pair an exact runtime (which may be infinite). This assumption basically corresponds to a "laboratory" setting for experiments, which abstracts from variables currently not under investigation.

[2]i.e. a distance function as used in the mathematical theory of metric spaces
[3]although also these cases can seamlessly be covered by choosing a discrete error function

Let $\mathcal{A}$ be a set of algorithms. To every algorithm $a \in \mathcal{A}$ we assign an IO-FUNCTION $f_a : \Omega \times \mathbb{R}^+ \rightarrow X \times 2$ with $2 := \{0, 1\}$. Hereby, $f_a(\omega, t) = (x, b)$ means that the algorithm a applied to the input (task, problem, ...) ω yields the result x after running time t together with the information whether the algorithm has already reached its final output ($b = 1$) or not yet ($b = 0$). As a natural constraint, we require f_a to additionally satisfy the condition that for all $t_2 \geq t_1$ we have that

$$f_a(\omega, t_1) = (x, 1) \text{ implies } f_a(\omega, t_2) = (x, 1),$$

i.e. after having indicated termination, the output of the algorithm (including the termination statement) will not change anymore. For convenience we write $f_a^{\text{res}}(\omega, t) = x$ and $f_a^{\text{term}}(\omega, t) = b$, if $f_a(\omega, t) = (x, b)$.

By $f_0 : \Omega \rightarrow X$ we denote the CORRECT OUTPUT FUNCTION, which is determined by some external specification or formal semantics of the problem. This enables us to verify the (level of) correctness of an answer $x \in X$ with respect to a particular input ω by determining $e(x, f_0(\omega))$ – the smaller the respective value, the better the answer. By our standing condition on e, $e(x, f_0(\omega)) = 0$ ensures $f_0(\omega) = x$ coinciding with the intuition.

To any algorithm a, we assign a RUNTIME FUNCTION $\varrho_a : \Omega \rightarrow \mathbb{R}^+_\infty$ by setting

$$\varrho_a(\omega) = \inf\{t \mid f_a^{\text{term}}(\omega, t) = 1\},$$

being the smallest time, at which the algorithm a applied to input ω indicates its termination.[4] Note that this implies $\varrho_a(\omega) = \infty$ whenever we have $f_a^{\text{term}}(\omega, t) = 0$ for all $t \in \mathbb{R}^+$. Algorithms, for which for all $\omega \in \Omega$ we have that $\varrho_a(\omega) < \infty$ and $f_a^{\text{res}}(\omega, t) = \perp$ for all $t < \varrho_a(\omega)$ are called ONE-ANSWER ALGORITHMS: They give only one output which is not $\perp$, and are guaranteed to terminate[5] in finite time.

Given an algorithm a, an input ω and a time unit t, we use the expression $e(a, \omega, t) = e(f_a^{\text{res}}(\omega, t), f_0(\omega))$ as a shorthand to denote the error of algorithm a on input ω at time t.

5.2.4 Defects

Clearly, for a given time t, the expression $e(f_a^{\text{res}}(\omega, t), f_0(\omega))$ provides a measure of how much the current result provided by the algorithm diverges from the correct solution. Moreover, it is quite straightforward to extend this notion to the whole input space (by taking into account the occurrence probability of the single inputs). This is done by the next definition.

The DEFECT $\delta(a, t)$ ASSOCIATED WITH AN ALGORITHM $a \in \mathcal{A}$ AT A TIME POINT t is given by

$$\delta : \mathcal{A} \times \mathbb{R}^+ \rightarrow \mathbb{R}^+_\infty : \delta(a, t) = E(e(f_a^{\text{res}}(\omega, t), f_0(\omega))) = \sum_{\omega \in \Omega} e(f_a^{\text{res}}(\omega, t), f_0(\omega)) P(\omega).$$

Note that E denotes the expected value, which is calculated by the rightmost formula.[6]

[4]We make the reasonable assumption that f_a^{res} is right-continuous.

[5]We impose termination here because our main interest is in reasoning with description logics for the Semantic Web. The same notion *without* imposing termination would also be reasonable, for other settings.

[6]The sum could easily be generalised to an integral – with P being a probability measure –, however it is reasonable to expect that Ω is discrete, and hence the sum suffices.

Furthermore, one can even abstract from the time and take the results after waiting "arbitrarily long": The (ULTIMATE) DEFECT of an algorithm $a \in \mathcal{A}$ is given by

$$\delta : \mathcal{A} \to \mathbb{R}^+_\infty : \delta(a) = \limsup_{t\to\infty} \delta(a,t).$$

We say that $a \in \mathcal{A}$ REALISES A DEFECTLESS APPROXIMATION if

$$\lim_{t\to\infty} \delta(a,t) = 0.$$

Note that $\delta(a) = 0$ in this case.

By the constraint put on the IO-function we get

$$\delta(a) = E(e(f_a^{\text{res}}(\omega, \varrho_a(\omega)), f_0(\omega))) = \sum_{\omega\in\Omega} e(f_a^{\text{res}}(\omega, \varrho_a(\omega)), f_0(\omega))P(\omega).$$

if a terminates for every input.

5.2.5 Comparing algorithms after termination

An algorithm a is EVERYWHERE MORE PRECISE THAN b iff $e(a,\omega,t) \le e(b,\omega,t)$ for any ω and t. This notion, which *guarantees* better precision on any input, is in practice too strong, since it makes many algorithms incomparable: on some inputs a is more precise than b, while on other inputs the reverse is true, ie. neither a strongly more precise than b nor the reverse. Hence, it makes sense to use the *expected* defect $\delta(a,t)$ instead of the input-specific error-function $e(a,\omega,t)$.

For $a, b \in \mathcal{A}$, we say that a is MORE PRECISE THAN b if it has smaller ultimate defect, i.e. if

$$\delta(a) \le \delta(b).$$

Furthermore, it is often interesting to have an estimate on the runtime of an algorithm. Again it is reasonable to incorporate the problems' probabilities into this consideration. So we define the AVERAGE RUNTIME[7] of algorithm a by

$$\alpha(a) = E(\varrho_a(\omega)) = \sum_{\omega\in\Omega} \varrho_a(\omega)P(\omega).$$

This justifies to say that a is QUICKER THAN b if

$$\alpha(a) \le \alpha(b).$$

Note that this does not mean that a terminates earlier than b on every input. Instead, it says that when calling the algorithm very often, the overall time when using a will be smaller than when using b – weighted by the importance of the input as measured by P.

Throughout the considerations made until here, it has become clear that there are two dimensions along which approximate reasoning algorithms can be assessed or compared: runtime behaviour and accuracy of the result. Clearly, an algorithm will be deemed better, if it outperforms another one with respect to the following criterion:

[7]We are aware that in some cases, it might be more informative to estimate the runtime behaviour via other statistical measures as e.g. the median.

Definition 1 *For $a, b \in \mathcal{A}$, we say that a* IS STRONGLY BETTER THAN *b if a is more precise than b and a is quicker than b.*

A more flexible definition than the one just introduced will be given later when we introduce the notion that an algorithm a is *better than* an algorithm b.

5.3 Extending the Framework to Anytime Algorithms

5.3.1 Properties of anytime algorithms

The definitions just given in Section 5.2.5 compare algorithms after termination, i.e. anytime behaviour of the algorithms is not considered. In order to look at anytime aspects, we need to consider the continuum of time points from initiating the anytime algorithm to its termination.

The term "anytime algorithm" was coined by Dean (Dean & Boddy, 1988) in the context of time dependent planning. Anytime algorithms expand upon the traditional view of a computational procedure as they offer to fulfill an entire spectrum of input-output specifications, over the full range of run-times, rather than just a single specification. An anytime algorithm is an implementation of a mapping from a set of inputs and time allocation into a set of outputs. For each input there is a corresponding set of possible outputs, each of which is associated with a particular time allocation and some measure of its quality. The advantage of this generalization is that computation can be interrupted at any time and still produce results of a certain quality, hence the name "anytime algorithm."

Zilberstein (Zilberstein, 1996) has identified a number of properties that characterise anytime algorithms. We will first informally discuss these properties. Where possible, we already provide a formalisation of the property using the apparatus presented above. In some cases, the formalisation must be postponed until we have introduced some more formal machinery below.

- **measurable:** being able to define $e(a, \omega, t)$ for any value of t and ω
- **recognisable:** being able to compute $e(a, \omega, t)$ efficiently at run-time for any value of t. Notice that this is stronger than *measurable.*
- **consistency:** same/similar quality for same resource usage. It amounts to demanding that $e(a, \omega, t)$ is functional in t for a given input ω.
- **monotonicity:** the defect $\delta(a, t)$ decreases monotonically as a function of t.
- **diminishing returns:** the defect decreases slower with increasing t, in other words $\delta(a, t)$ is concave
- **interuptability:** $f_a(\omega, t)$ can be calculated for every value of t on any input ω.
- **defectless approximation:** a is a ($\lim_{t\to\infty}$d(a,t)$= 0$).

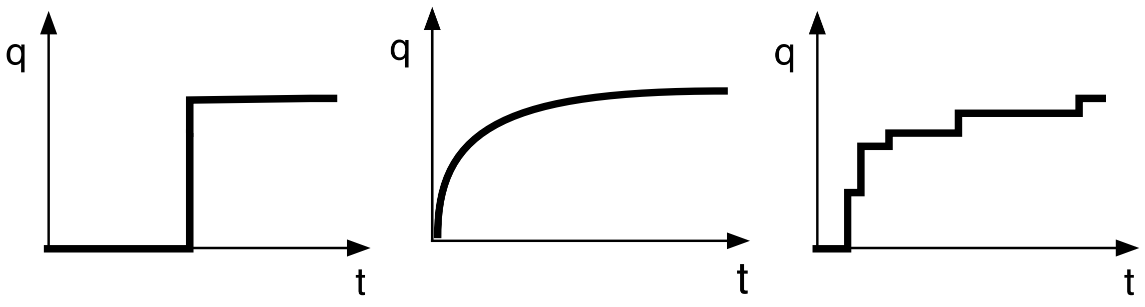
There is no agreement in the literature on which of these properties are *required* in order for an algorithm to be called anytime. A minimal option would be to only require that the system monotonically approaches perfection in the limit, as follows:

Definition 2 *We say that an algorithm $a \in \mathcal{A}$ is an* ANYTIME ALGORITHM *if it is interuptable and recognisable.*

The other properties are desirable: monotonicity guarantees that results will only improve over time and never degrade; consistency guarantees reliable behaviour; diminishing returns guarantee the greatest gains early in the process; and defectlessness guarantees convergence to a correct answer. However, none of these properties are *required* for an algorithm to be anytime.

5.3.2 Performance profiles

The first to introduce the notion of performance profiles for anytime algorithms was Zilberstein (Zilberstein & Russell, 1996). The performance profile of an algorithm characterizes the quality of its output as a function of computation time. All algorithms "whether standard or anytime" have a performance profile. Figure 5.1 shows typical performance profiles for standard algorithms 5.3a and idealized anytime algorithms 5.3b. The performance profile of the standard algorithm shows that no results are available until its termination at which point the exact result is returned. The idealized anytime algorithm provides output whose quality improves gradually over time. In practice, the improvement in quality of an anytime algorithm may look more like the profile shown in Figure 5.2c.



(a) Standard algorithm (b) Idealized anytime algorithm (c) Actual anytime algorithm

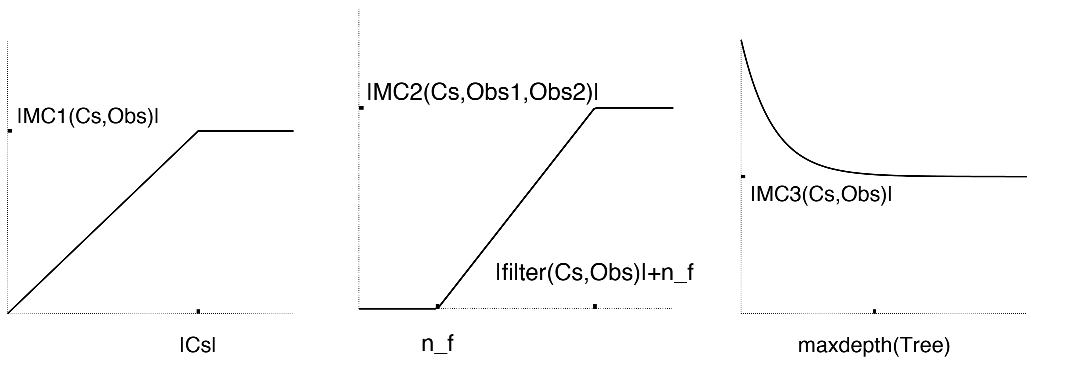
Figure 5.1: Typical performance profiles.

input-specific performance profile The simplest performance profiles plot the anytime behaviour of an algorithm a on a particular input ω. This corresponds exactly to plotting the value of $1/e(a, \omega, t)$ as a function of t. (we plot $1/e(a, \omega, t)$ instead of simpy $e(a, \omega, t)$ simply to conform to the tradition that performance profiles show the increase of the quality, and not the reduction of the error.

expected performance profile The input-specific performance profiles are only defined on a given input ω. Expected performance profiles capture the anytime behaviour of the algorithm over the entire input space, which corresponds precisely to plotting $1/\delta(a, t)$ as a function of t (again plotting increase in quality instead of decrease of defect). In practice it will of course not be possible to compute $\delta(a, t)$ precisely: computing $\delta(a, t)$ requires knowing the output error-value $e(a, \omega, t)$ for all possible inputs ω.

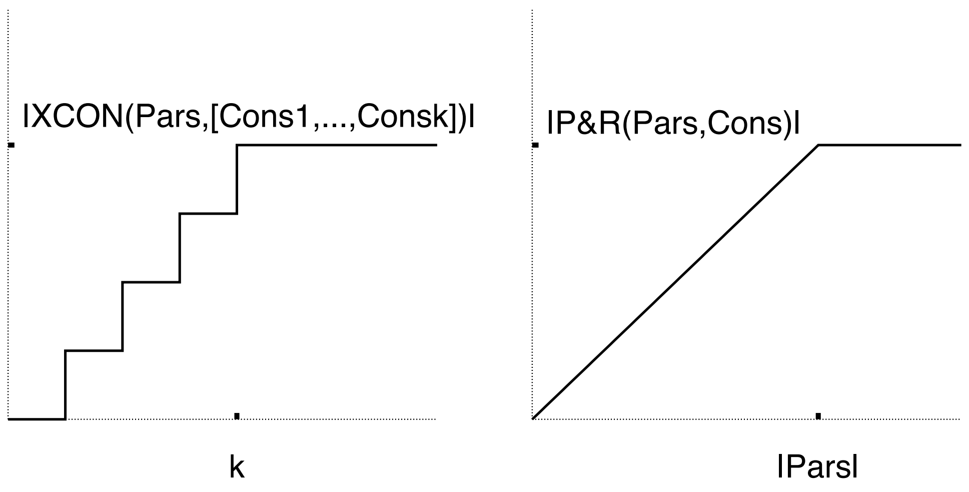
Typically, these error-value values cannot be determined analytically, but must be observed empirically. hence computing $\delta(a, t)$ would require running a on all values $\omega \in \Omega$! Instead, expected performance profiles are typically constructed empirically by collecting statistics on the performance of an algorithm over many input instances. Figure 7 in (Zilberstein & Russell, 1996) shows the result of sampling the error-value over many inputs and many time-points for a particular algorithm and Figure 8 (Zilberstein & Russell, 1996) shows the resulting expected performance profile.

Sometimes it is possible to analytically determine the *qualitative shape* of an expected performance profile. This then allows a plot of the shape of the curve, without pinning the curve down quantitatively. Figures 5.2 and 5.3 show the qualitative performance profiles resulting from the qualitative analysis of a number of different reasoning algorithms (Harmelen & Teije, n.d.).



(a) Standard algorithm (b) Idealized anytime algorithm (c) Actual anytime algorithm

Figure 5.2: Performance profile of MC1, MC2 and MC3.



(a) Standard algorithm (b) Idealized anytime algorithm

Figure 5.3: Performance profile of XCON and P&R.

Zilberstein (Zilberstein & Russell, 1996) also discusses conditional performance profiles, which not only plot $\delta(a, t)$ as a function of t, but also a function of the input quality. In this paper we do not consider the possibility of variable input quality, altough this is would lead to an interesting of robustness and stability of approximation algorithms in the face of approximate *input*.

dynamic performance profiles . (Hansen & Zilberstein, 1996) proposed the notion of a dynamic performance profile. It follows the observation that quality improvement can better be predicted at run time when the quality of the currently available result is taken into account.

A dynamic performance profile of an anytime algorithm plots the probability of getting a particular increase in quality by resuming the algorithm for time interval Δt when the currently available solution has quality q. Such profiles are particularly useful during at runtime, when an anytime algorithm has to decide if it's useful to spend an additional amount of time Δt, based on the prediction of the gain in quality.

5.3.3 Comparing Anytime Algorithms

For $a, b \in \mathcal{A}$, we say that a is MORE PRECISE THAN b AT TIME POINT t if it has smaller defect wrt. a and t, i.e. if

$$\delta(a,t) \leq \delta(b,t).$$

Obviously, is is reasonable to say about two algorithms a and b – be they anytime or not –, that (1) a is better than b if a is more precise than b at any time point. This is called "a dominates b" in (Zilberstein & Russell, 1996). A less strict – and apparently more reasonable – requirement accumulates the difference between a and b over the entire runtime, stating that (2) a is better than b if $\sum_{\omega\in\Omega} P(\omega) \int_{t=0}^{\max\{\varrho_a(\omega),\varrho_b(\omega)\}} (e(f_a^{\text{res}}(\omega,t), f_0(\omega)) - e(f_b^{\text{res}}(\omega,t), f_0(\omega))\mathrm{d}t \leq 0$. We find formula (2) still not satisfactory as it ignores the reasonable assumption that some time points might be more important than others, i.e. they need to be weighted more strongly. Formally, this is done by using a different measure for the integral or – equivalently – a density function $\bar{f} : \mathbb{R}^+ \to \mathbb{R}^+$, which modifies the integral. Summarizing, we now define for two (not necessarily anytime) algorithms a and b that (3) a IS BETTER THAN b (wrt. a given density function $\bar{f}$) if

$$\sum_{\omega\in\Omega} P(\omega) \int_{t=0}^{\max\{\varrho_a(\omega),\varrho_b(\omega)\}} \big(e(f_a^{\text{res}}(\omega,t), f_0(\omega)) - e(f_b^{\text{res}}(\omega,t), f_0(\omega))\big)\bar{f}(t)\mathrm{d}t \leq 0.$$

Notice that this definition favours algorithms with early decreasing defect values, which corresponds to the above mentioned desirable property of diminishing returns.

Notice also that this definition does *not* favour algorithms with a low run-time.

Our definition (3) specialises to the case in (2) for the constant density function $\bar{f} \equiv 1$. We cannot capture (1) with our definition by one specific choice of $\bar{f}$, so in the case of (1) we simply say that a is more precise than b at any time point.[8]

[8]However, (1) could be formulated in terms of (3) as a being better than b for all Dirac delta functions that have their singularity at a nonnegative place.

Clearly, the choice of the density function depends on the considered scenario. In cases where only a fixed time t_{timeout} can be waited before a decision has to be made based on the results acquired so far, the value $\bar{f}(t)$ of density function would be set to zero for all $t \geq t_{\text{timeout}}$. Usually earlier results are preferred to later ones which would justify the choice of an $\bar{f}$ that is monotonically decreasing.

5.4 Anytime Algorithms by Composition

Realised approximate reasoning systems are often not anytime. However, it is possible to obtain anytime behaviour by composing one-answer algorithms.

Assume that a number of algorithms a_i $(i = 1, \ldots, n)$ is given. Furthermore, assume there is an ORACLE ALGORITHM **c** whose behaviour can be described by a function $c : (X \times \mathtt{2})^n \to X \times \mathtt{2}$ which combines a vector of outputs $(a_1(\omega, t), \ldots, a_n(\omega, t))$ of the algorithms a_i and yields a single output. Given an input ω, the invocation of all a_i in parallel and the subsequent call of the oracle algorithm yield a new algorithm $c_{a_1,\ldots,a_n}$ with IO-function

$$f_{c_{a_1,\ldots,a_n}}(\omega, t) = c(a_1(\omega, t), \ldots, a_n(\omega, t)).$$

The definition just given is very general in order to allow for a very free combination, depending on the algorithms which are being combined. For the general setting, we impose only the very general constraint that for all $x_1, \ldots, x_n \in X$ we have

$$c((x_1, 1), \ldots, (x_n, 1)) = (x, 1)$$

for some x, and also that the natural constraint from page 34 on the corresponding IO-function $f_{c_{a_1,\ldots,a_n}}$ is satisfied. This is just to ensure $\varrho_{c_{a_1,\ldots,a_n}}(\omega) \leq \max\{\varrho_{a_1}, \ldots, \varrho_{a_n}\}$, i.e. the "combiner" indicates termination at the latest whenever all of the single input algorithms a_i do so.

It is more interesting to look at more concrete instances of oracles. Assume now that $a_1, \ldots, a_{n-1}$ are one-answer algorithms and that a_n is an (always terminating) sound and complete algorithm. Let **c** be such that

$$c(a_1(\omega, \varrho_{a_n}(\omega)), \ldots, a_{n-1}(\omega, \varrho_{a_n}(\omega)), a_n(\omega, \varrho_{a_n}(\omega))) = (f_{a_n}^{\text{res}}(\omega), 1).$$

Then it is easy to see that $c_{a_1,\ldots,a_n}$ is anytime.

If we know about soundness or completeness properties of the algorithms $a_1, \ldots, a_{n-1}$, then it is also possible to guarantee that $c_{a_1,\ldots,a_n}$ is monotonic anytime. This can be achieved in several ways, and we give one specific example based on ABox reasoning in description logics:

Assume that each input consist of a class description C over some description logic L, and each output consists of a set of (named) individuals. For constructing an oracle from such algorithms, we will actually consider as outputs *pairs* (A, B) of sets of individuals. Intuitively, A contains only individuals which are *known* to belong to the extension of C, while B constitutes an individual set which *is known to contain* all individuals in the extension of C. A single output (set) A can be equated with the output pair (A, A). Now let $a_1, \ldots, a_n$ be sound[9] but

[9]We mean soundness in the following sense: If the set I of individuals is the correct answer, then the algorithms yields as output a pair (A, A) of sets with $A \subseteq I$.

incomplete[10] one-answer algorithms over L, let $b_1, \dots, b_m$ be complete but unsound one-answer algorithms over L and let a be a sound, complete and terminating algorithm over L, i.e. $f_a^{\text{res}}(C, \varrho_a)$ – which we denote by C_a – contains exactly all named individuals that are in the extension of C as a logical consequence of the given knowledge base. Under this assumption, we know that $f_{a_i}^{\text{res}}(C, \varrho_{a_i}) = (C_{a_i}, \mathbf{I})$ and $f_{b_j}^{\text{res}}(C, \varrho_{b_j}) = (\emptyset, C_{b_j})$ for some sets C_{a_i} and C_{b_j}, where I stands for the set of all (known) individuals, and furthermore we know that $C_{a_i} \subseteq C_a \subseteq C_{b_j}$ for all i, j.

The oracle $\mathbf{c}$ is now defined as follows.

$$\begin{aligned}
&\mathbf{c}(a_1(C,t), \dots, a_n(C,t), b_1(C,t), \dots, b_m(C,t), a(C,t)) \\
&= \begin{cases}
((f_a^{\text{res}}(C,t), f_a^{\text{res}}(C,t)), 1) & \text{for } t \geq \varrho_a(C), \\
((\mathit{upper}, \mathit{lower}), \mathit{term}) & \text{for } t < \varrho_a(C) \\
\quad \text{where } \mathit{lower} = \bigcup_{(A_i, B_i, 1) = f_{a_i}(C,t)} A_i, & \\
\qquad \mathit{upper} = \bigcap_{(A_j, B_j, 1) = f_{b_j}(C,t)} B_j, & \\
\qquad \mathit{term} = 1 \text{ if } \mathit{lower} = \mathit{upper}, \text{ otherwise } 0. &
\end{cases}
\end{aligned}$$

Note that the empty set union is by definition the empty set, while the empty set intersection is by definition I.

In words, the oracle realises the following behaviour: if the sound and complete subalgorithm has terminated, display its result. Before, use the lower resp. upper bounds delivered by the sound resp. complete algorithms to calculate one intermediate lower and one intermediate upper bound. If those two happen to coincide, the correct result has been found and may terminate without waiting for a's termination. This *squeezing in* of the correct result now also explains why we have chosen to work with pairs of sets as outputs.

As error function, we might use the sum of the symmetric difference between A and A_0, respectively between B and A_0, i.e.

$$e((A, B), (A_0, A_0)) = |A_0 \setminus A| + |B \setminus A_0|.$$

We could also use a value constructed from similar intuitions like precision and recall in information retrieval, but for our simple example, this error function suffices. It is indeed now straightforward to see that $\mathbf{c}_{a_1,\dots,a_n,b_1,\dots,b_m,a}$ is monotonic anytime. It is also clear that $\mathbf{c}_{a_1,\dots,a_n,b_1,\dots,b_m,a}$ is more precise than any of the a_i and b_j, at all time points.

5.5 An Example

In this section, we will instantiate the very general framework established in the preceding sections. We will use the presented techniques to compare three approximate reasoning algorithms and compose a (simple) anytime algorithm following the example at the end of Section 5.4.

Consider the three algorithms Screech-all, Screech-none and KAON2, as discussed in (Tserendorj, Rudolph, Krötzsch, & Hitzler, 2008). We do not intend

[10]We mean completeness in the following sense: If the set I of individuals is the correct answer, then the algorithms yields as output a pair (A, A) of sets with $I \subseteq A$.

to give any details here, and it shall suffice to mention that these are one-answer algorithms for reasoning with the description logic $\mathcal{SHIQ}$, and the task considered is instance retrieval for named classes. Screech-all is complete but unsound, Screech-none is sound but incomplete, and KAON2 is sound and complete.

Following the general framework, we first have to stipulate the probability space (Ω, P) for our case. Here we introduce the first simplifying assumptions, which are admittedly arguable, but will suffice for the example:

- We consider only one knowledge base, namely the well-known Wine ontology. Further evaluation data is available (Tserendorj et al., 2008) but will not be taken into account for the illustrating example.
- As queries, we consider only instance retrieval tasks, i.e. given an atomic class description, we query for the set of individuals which can be inferred to be instances of that class. Hence Ω – the query space – consists of named classes $\mathbf{C}$ of the Wine ontology the instances of which are to be retrieved: $\Omega = \mathbf{C}$. Examples for named classes in this ontology are e.g. `Chardonnay`, `StEmilion` or `Grape`.
- All those instance retrieval queries $\omega \in \Omega$ are assumed to be equally probable to be asked to the system, hence

  $$P(\omega) = \frac{1}{|\mathbf{C}|} \text{ for all } \omega \in \Omega.$$

  Obviously, the probability of a query could also be assumed differently, e.g. correlating with the number of instances the respective class has. Nevertheless, for the sake of simplicity we will stick to the equidistributional approach.

Obviously, the output space X consists of subsets of the set of individuals $\mathbf{I}$ from the Wine ontology together with the no-output symbol $\bot$: $X = 2^{\mathbf{I}} \cup \{\bot\}$. As the error function e comparing an algorithm's output I with the correct one I_0, we use the inverted value of the common f-measure, i.e.

$$e(I, I_0) := 1 - \frac{2 \cdot \mathit{precision} \cdot \mathit{recall}}{\mathit{precision} + \mathit{recall}}$$

where (as usual)

$$\mathit{precision} := \frac{|I \cap I_0|}{|I|} \qquad \text{and} \qquad \mathit{recall} := \frac{|I \cap I_0|}{|I_0|}.$$

According to the proposed handling of $\bot$, we stipulate the overall "worst-case distance": $e(\bot, I_0) = 1$ for all $I \subseteq \mathbf{I}$.

As mentioned before, the set $\mathcal{A}$ of considered algorithms comprises three items:

$$\mathcal{A} = \{\text{KAON2, Screech-all, Screech-none}\}$$

For every of those algorithms we carried out comprehensive evaluations: we queried for the class extensions of every named class and stored the results as well as the time needed. By their nature none of the considered algorithms exhibits a genuine anytime behavior, however, instead of displaying the "honest" $\bot$ during their calculation period, they could be made to display an arbitrary intermediate result. It is straightforward to choose the empty set in order to obtain better results: most class extensions will be by far smaller than half of the individual set, hence the distance of the correct result to the empty set will be a rather good guess.

Hence, for any algorithm a of the above three and any class name C let I_C denote be the set of retrieved instances and t_C denote the measured runtime for accomplishing this task. Then we can define the IO-function as

$$f_a(C,t) = \begin{cases} (\emptyset, 0) \text{ if } t < t_C \\ (I_C, 1) \text{ otherwise.} \end{cases}$$

The values of the correct output function f_0 can be found via KAON2, as this algorithm is known to be sound and complete. Moreover, the runtime functions $\rho_a(C)$ of course coincide in our case with the runtimes t_C measured in the first place. Since all of the considered algorithms are known to terminate, no ρ_a will ever take the value ∞.

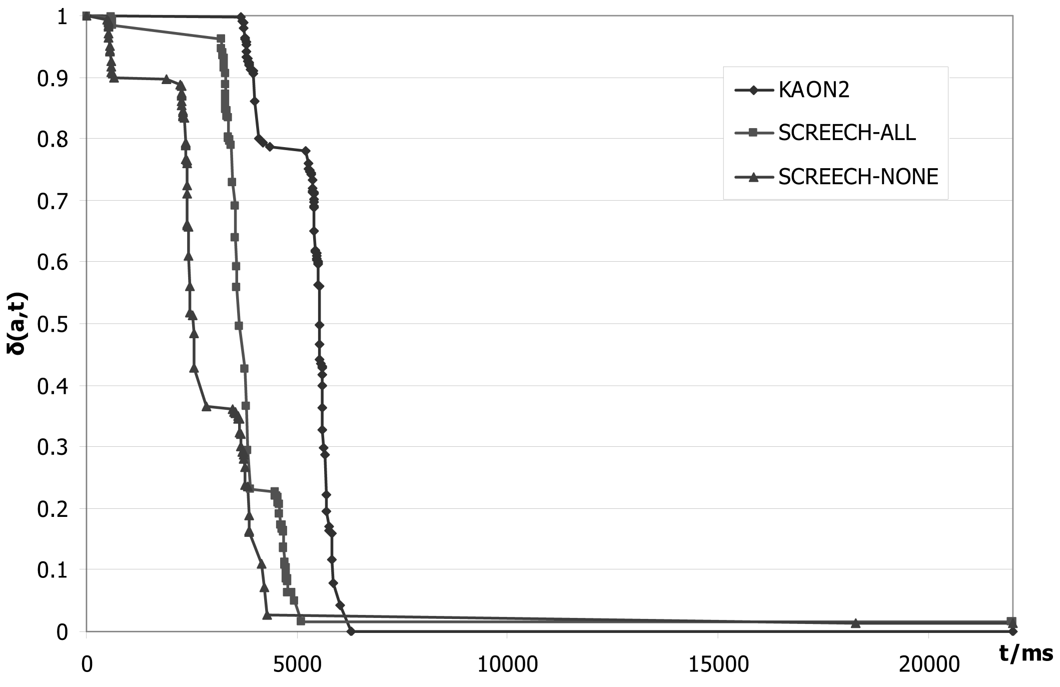


Figure 5.4: Defect over time.

After this preconsiderations, we are ready to carry out some calculations estimating the quality of the considered algorithms. Figure 5.4 shows a plot depicting the decrease of the defect for all the three algorithms. As expected, there is an ultimate defect for the two screech variants, namely 0.013 for Screech-none and 0.015 for Screech-all, i.e. with respect to the terminology introduced earlier, we can say that Screech-none is more precise than Screech-all. While the defect of KAON2 is initially greater than those of the screech variants, it becomes better than them at about 6 seconds and decreases to zero defect after about 7 seconds. In other words, Screech-all is more precise than KAON2 at all time points less than 6 seconds. A first conclusion from this would be: if a user is willing to wait for 7 seconds for an answer (which then would be guaranteed to be correct) KAON2 would be the best choice, otherwise (if time is crucial and precision not), screech-all might be a better choice as it shows the quickest defect decrease.

If we now assume a time-critical application where responses coming in later than, say, 5 seconds are ignored, we can describe this by the fact that SCREECH-ALL is better than KAON2 with respect to the density function

$$\bar{f}(x) = \begin{cases} 1 & 0 \leq x \leq 5, \\ 0 & \text{otherwise.} \end{cases}$$

Considering the fact that SCREECH-ALL is complete, SCREECH-NONE is sound, and KAON2 is both, we can now utilize a variant of the oracle given in the example from Section 5.4. The behaviour of the combined algorithm can in this simple case be described as follows. It indicates termination whenever one of the following occurs:

- KAON2 has terminated. Then the KAON2 result is displayed as solution.
- Both SCREECH-ALL and SCREECH-NONE have terminated with the same result. In this case, the common result will be displayed as the final one.

If none of above is the case, the experimental findings suggest to choose the SCREECH-NONE result as intermediate figure. The algorithm obtained that way is anytime and more (or equally) precise than any of the single algorithms at all time points.

5.6 Summary

Approaches to approximate reasoning tackle the problem of scalability of deducing implicit knowledge. Especially if this is done on the basis of large-scale knowledge bases or even the whole Web, often the restriction to 100% correctness has to be abandoned for complexity reasons, in particular if quick answers to posed questions are required. Anytime algorithms try to fulfill both needs (speed and correctness) by providing intermediate results during runtime and continually refining them.

In this chapter, we have provided solid mathematical foundations for the assessment and comparison of approximate reasoning algorithms with respect to correctness, runtime and anytime behaviour. We are confident that this general framework can serve as a means to classify algorithms w.r.t. their respective characteristics and help in deciding which algorithm best matches the demands of a concrete reasoning scenario.

As opposed to our example scenario, in most practical cases, it will be unfeasible or even impossible to measure the whole input space as it will be too large or even infinite. That is where statistical considerations come into play: one has to identify and measure representative samples of the input space. The first part of this is far from trivial: for fixed settings with frequently queried knowledge bases, such a sample could be determined by protocolling the actually posed queries over a certain period of time. Another way would be to very roughly estimate a distribution based on plausible arguments. Respective heuristics would be: (1) the more complex a query the more unlikely, (2) queries of similar structure are similarly frequent resp. likely, (3) due to some bias in human conceptual thinking, certain logical connectives (e.g. conjunction) are preferred to others (e.g. disjunction, negation) which also gives an opportunity to estimate a query's frequency based on the connectives it contains. Admittedly, those heuristics are still rather vague and more thorough research is needed to improve reliability of such estimates.

In general, the proposed intelligent combination of several algorithms with different soundness/completeness properties (as well as being specialised to certain logical fragments) can increase speed and might help avoid heavy-weight reasoning in cases. We are confident, that this idea can be easily generalised to reasoning tasks other than instance retrieval. Obviously, this strategy comes with an immediate opportunity of parallelisation even if the single algorithms have to be treated as black boxes. Hence, this approach could also be conceived as a somewhat exotic approach to distributed reasoning.

6 Conclusion

In this deliverable, the first of a series three documents concerned with the definition of a Framework for Measuring and Evaluating Heuristic Problem Solving, we have made the first steps towards defining such framework by considering the theoretical foundations of the theory of evaluation, discussing several important aspects related to the process of evaluating LarKC and its platform and, reporting on several dimensions and methods by which the components of the platform can be evaluated.

We started the document by providing a clarification of the deliverable's main concepts and an overview over the specific challenges presented by the evaluation of the LarKC project and platform. This chapter clarifies the notions of *measurement*, *evaluation* and *heuristic problem solving*—both in general and in the specific context of the LarKC project. We have also identified and discussed an initial set of quantitative dimensions that we think need to be measured in order to evaluate the performance of the LarKC platform. Such metrics include robustness, speed, accuracy, scalability, usability and user satisfaction and quality and impact. These metrics can be evaluated at different levels such as at component (plug-in) level and at workflow level, among others. Furthermore, and in regard to heuristic problem solving we have identified several usages and outlined three ways in which the concept can be applied to LarKC: as methods used in plug-in design, as stopping rules implemented in deciders, and as workflows that are configured to solve particular problems. The evaluation of heuristic problem solving follows the same methods and principles as evaluation in general. We also tackle the aspect of user feedback, the role of users in the evaluation of the LarKC project and platform and mentioned some approaches to collecting evaluative user feedback, like for example to automatically prompt users to submit quality ratings upon the completion of a user query. The document also discusses the notion of heuristic problem solving and the multiple ways in which heuristics are embedded in LarKC, namely as problem-solving strategies used at the plug-in level, which in turn may lead to heuristic-based plug-ins, and at the workflow level, which leads to the notion of workflows as heuristic problem solvers. We highlight three methodologies that are particularly relevant for the evaluation of heuristic problem solving, namely *comparison with alternative methods*, *user feedback* and *cross-validation*.

In terms of the evaluation of the different plug-in types the document reports on methods and criteria that we think could be used for evaluating the performance of plug-ins. More concretely and in relation to the evaluation of Identify plug-ins we have considered the speed of an identify plug-in and its accuracy, defined in terms of the notions of precision and recall, which are widely used in the field of Information Retrieval. In addition to this, and in accordance to basic notions of evaluation theory, we proposed to conduct both intrinsic and extrinsic evaluation of identify plug-ins; for extrinsic evaluation we have also reported on the plans to evaluate this type of plug-in in the context of two LarKC use cases, namely WP7a/b.

Concerning the evaluation of the performance of Transform plug-ins we have given an overview of the challenges that need to be addressed when evaluating transform plug-ins that borrow techniques from the area of Machine Learning. We have also discussed rank list evaluation.

In terms of the evaluation of Reason plug-ins we have laid down the foundations for a statistical approach to evaluating approximate and anytime reasoning algorithms. We have done that in a very abstract manner, which can be made concrete in different ways, depending on the use case being considered. At the same time, we have used this statistical perspective to precisely define approximate reasoning notions which to date have remained quite vague. Furthermore, we have shown that our mathematical modeling can be used for guiding the development of composed approximate reasoning systems. In the end, our mathematical modeling can be used for rigorous comparative statistical evaluation of approximate reasoning algorithms. Moreover, we have proposed to assess the usefulness of an approximate reasoning algorithm by looking at two aspects or dimensions: Runtime behaviour and accuracy of results. By introducing an *error function* that gives a quantitative measure of the extent to which an output deviates from the desired output (as computed by a sound and complete reasoner), we are able to formalize the fact that out of two wrong answers one might still be better than the other since it is "closer" to the correct result.

As for future work the main challenge ahead is how to integrate the different evaluation criteria into a coherent and well-founded framework that enables the evaluation of the LarKC platform at different levels and supports users in deciding which particular combination of plug-ins to use in order to solve a specific problem. We also need to specify additional qualitative and quantitative dimensions that can be used for evaluating specific types of plug-ins as well as evaluating the performance of the entire platform. Furthermore, we need to investigate how such framework could be implemented in such a way as to support automatic workflow configuration and execution, a task carried out by a Decider plug-in, and investigate how such plug-in can use the framework to monitor the performance of the platform.

References

Cadoli, M., & Scarcello, F. (2000, may). Semantical and computational aspects of Horn approximations. *Artificial Intelligence*, *119*(1).

Cadoli, M., & Schaerf, M. (1995). Approximate inference in default reasoning and circumscription. *Fundamenta Informaticae*, *23*, 123–143.

Czerlinski, J., Gigerenzer, G., & Goldstein, D. (1999). How good are simple heuristics. In G. Gigerenzer, P. M. Todd, & the ABC research group (Eds.), *Simple heuristics that make us smart* (pp. 97–118). New York, NY, USA: Oxford University Press.

Dalal, M. (1998). Anytime clausal reasoning. *Annals of Mathematics and Artificial Intelligence*, *22*(3–4), 297–318.

Dean, T., & Boddy, M. (1988, August). An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)* (pp. 49–54). AAAI Press/MIT Press.

Dowling, W. P., & Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, *1*, 267–284.

Fensel, D., Harmelen, F. van, Andersson, B., Brennan, P., Cunningham, H., Valle, E. D., et al. (2008). Towards LarKC: a platform for web-scale reasoning.

In *Proceedings of the ieee international conference on semantic computing (icsc 2008), august 4-7, 2008, santa clara, ca, usa.* Los Alamitos, CA, USA: IEEE Computer Society Press. Available from `http://www.larkc.eu/wp-content/uploads/2008/05/larkc-icsc08.pdf`

Fensel, D., & Harmelen, F. V. (2007, March/April). Unifying reasoning and search to web scale. *IEEE Internet Computing*, *11*(2), 94–96.

Gigerenzer, G. (2000). *Adaptive thinking: Rationality in the real world.* New York, NY: Oxford University Press.

Gigerenzer, G. (2008). *Rationality for mortals: Risk and rules of thumb.* New York, NY: Oxford University Press.

Gigerenzer, G., & Brighton, H. (2009). Homo heuristicus: Why biased minds make better inferences. *Topics in Cognitive Science*, *1*(1), 107–143.

Gigerenzer, G., & Goldstein, D. G. (1996). Reasoning the fast and frugal way: Models of bounded rationality. *Psychological Review*, *103*, 650–669.

Gigerenzer, G., & Selten, R. (2001). *Bounded rationality: The adaptive toolbox.* Boston, MA: MIT Press.

Gigerenzer, G., Todd, P. M., & the ABC research group. (1999). *Simple heuristics that make us smart.* New York, NY, USA: Oxford University Press.

Goldstein, D. G., & Gigerenzer, G. (2002). Models of Ecological Rationality: The Recognition Heuristic. *Psychological Review*, *109*(1), 75–90.

Groot, P., Stuckenschmidt, H., & Wache, H. (2005). Approximating description logic classification for semantic web reasoning. In A. Gómez-Pérez & J. Euzenat (Eds.), *The semantic web: Research and applications, second european semantic web conference, eswc 2005, heraklion, crete, greece, may 29 - june 1, 2005, proceedings* (Vol. 3532, p. 318-332). Springer.

Groot, P., Teije, A. ten, & Harmelen, F. van. (2004, June). Towards a structured analysis of approximate problem solving: a case study in classification. In *Proceedings of the ninth international conference on principles of knowledge representation and reasoning (KR'04).* Whistler, Colorado.

Hansen, E. A., & Zilberstein, S. (1996). Monitoring the progress of anytime problem-solving. In *Aaai/iaai* (Vol. 2, p. 1229-1234).

Harmelen, F. van, & Teije, A. ten. (n.d.). Describing problem solving methods using anytime performance profiles booktitle =.

Harmelen, F. van, & Teije, A. ten. (2000, August). Describing problem solving methods using anytime performance profiles. In *Proceedings of ECAI'00* (pp. 181–186). Berlin.

Hitzler, P., & Vrandecic, D. (2005). Resolution-based approximate reasoning for OWL DL. In Y. Gil et al. (Eds.), *Proceedings of the 4th international semantic web conference, galway, ireland, november 2005* (Vol. 3729, p. 383-397). Springer, Berlin.

Holger Wache, H. S., Perry Groot. (2005). Scalable Instance Retrieval for the Semantic Web by Approximation. In (pp. 245–254).

Horrocks, I., Li, L., Turi, D., & Bechhofer, S. (2004). The Instance Store: DL reasoning with large numbers of individuals. In *Proceedings of the international workshop on description logics, dl2004, whistler, canada* (pp. 31–40).

Horvitz, E. J. (1987). Reasoning about beliefs and actions under computational resource constraints. In L. N. Kanal, T. S. Levitt, & J. F. Lemmer (Eds.), *Uncertainty in artificial intelligence 3* (pp. 301–324). Amsterdam, The Netherlands: Elsevier.

Jarvelin, K., & Kekalainen, J. (2000). IR evaluation methods for retrieving highly relevant documents. In *Sigir'00*.

Martignon, L., & Hoffrage, U. (1999). Why does one-reason decision making work? A case study in ecological rationality. In G. Gigerenzer, P. M. Todd, & the ABC research group (Eds.), *Simple heuristics that make us smart* (pp. 119–140). New York, NY, USA: Oxford University Press.

Martignon, L., & Hoffrage, U. (2002). Fast, frugal, and fit: Simple heuristics for paired comparison. *Theory and Decision*, *52*(1), 29–71.

Michalewicz, Z., & Fogel, D. B. (2004). *How to solve it: Modern heuristics* (Second Edition ed.). New York, NY, USA: Springer-Verlag.

Pan, J. Z., & Thomas, E. (2007). Approximating OWL-DL ontologies. In *Proceedings of the twenty-second aaai conference on artificial intelligence, july 22-26, 2007, vancouver, british columbia, canada* (pp. 1434–1439). AAAI Press.

Patton, M. Q. (1996). A world larger than formative and summative. *American Journal of Evaluation*, *17*(2), 131–144.

Perry Groot, H. W., Heiner Stuckenschmidt. (2005). Approximating Description Logic Classification for Semantic Web Reasoning.

Polya, G. (1947). *How to solve it: A new aspect of mathematical method.* Princeton, NJ, USA: Princeton University Press.

Schaerf, M., & Cadoli, M. (1995). Tractable reasoning via approximation. *Artificial Intelligence*, *74*, 249-310.

Scriven, M. (1991). Beyond formative and summative evaluation. In M. W. McLaughlin & D. C. Phillips (Eds.), *Evaluation and education: At quarter century* (pp. 18–64). Chicago, IL: The University of Chicago Press.

Selman, B., & Kautz, H. A. (1991). Knowledge compilation using Horn approximations. In *Proceedings of the ninth national conference on artificial intelligence (AAAI-91)* (pp. 904–909).

Shah, A. K., & Oppenheimer, D. M. (2008). Heuristics made easy: An effort-reduction framework. *Psychological Bulletin*, *134*(2), 207–222.

Simon, H. A. (1990). Invariants of human behavior. *Annual Reviews in Psychology*, *41*(1), 1–20.

Stockmann, R. (2000). Evaluation in Deutschland. In R. Stockmann (Ed.), *Evaluationsforschung. Grundlagen und ausgewählte Forschungsfelder* (pp. 11–40). Opladen: Leske + Budrich.

Stuckenschmidt, H. (2007). Partial matchmaking using approximate subsumption. In *Proceedings of the twenty-second aaai conference on artificial intelligence, july 22-26, 2007, vancouver, british columbia, canada* (pp. 1459–1464). AAAI Press.

Stuckenschmidt, H., & Harmelen, F. van. (2002). Approximating terminological queries. In H. Larsen & et al (Eds.), *Proc. of the 4th international conference on flexible query answering systems (FQAS)'02).* Springer.

Tresp, V., Huang, Y., Bundschus, M., & Rettinger, A. (2009). Materializing and querying learned knowledge. In *Eswc workshop on inductive reasoning and machine learning on the semantic web (irmles 2009).*

Tserendorj, T., Rudolph, S., Krötzsch, M., & Hitzler, P. (2008). Approximate OWL reasoning with Screech. In *Proceedings of the 2nd international confer-*

ence on web reasoning and rule systems, rr2008, karlsruhe, germany, october 2008. (To appear)

Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science*, *185*(4157), 1124–1131.

Zanakis, S. H., & Evans, J. R. (1981). Heuristic "optimization": Why, when, and how to use it. *Interfaces*, *11*(5), 84–91. Available from `http://www.jstor.org/stable/25060151`

Zhong, N., Yao, Y., Qin, Y., Lu, S., Hu, J., & Zhou, H. (2008). Towards granular reasoning on the web. In *Proceedings of the 2008 workshop on new forms of reasoning for the semantic web: Scalable, tolerant and dynamic (NEFORS 2008).* Bangkok, Thailand: 3rd Asian Semantic Web Conference (ASWC 2008).

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, *17*(3), 73-83.

Zilberstein, S., & Russell, S. (1996). Optimal composition of real-time systems. *Artificial Intelligence*, *82*, 181–213.