What Is Approximate Reasoning?*

Sebastian Rudolph[1], Tuvshintur Tserendorj[2], and Pascal Hitzler[1]

[1] AIFB, University of Karlsruhe, Germany
[2] FZI Karlsruhe, Germany

Abstract. Approximate reasoning for the Semantic Web is based on the idea of sacrificing soundness or completeness for a significant speed-up of reasoning. This is to be done in such a way that the number of introduced mistakes is at least outweighed by the obtained speed-up. When pursuing such approximate reasoning approaches, however, it is important to be critical not only about appropriate application domains, but also about the quality of the resulting approximate reasoning procedures. With different approximate reasoning algorithms discussed and developed in the literature, it needs to be clarified how these approaches can be compared, i.e. what it means that one approximate reasoning approach is better than some other. In this paper, we will formally define such a foundation for approximate reasoning research. We will clarify – by means of notions from statistics – how different approximate algorithms can be compared, and ground the most fundamental notions in the field formally. We will also exemplify what a corresponding statistical comparison of algorithms would look like.

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.

Introducing approximate reasoning in the Semantic Web field is motivated by the following observation: most nowadays' 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

* Research reported in this paper was partially supported by the EU in the IST project NeOn (IST-2006-027595, `http://www.neon-project.org/`)), by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project, and by the German Federal Ministry of Education and Research (BMBF) under the Theseus project, `http://theseus-programm.de`.

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.

In many cases, however, the time costs will be the most critical ones, as a user will not be willing to wait arbitrarily long for an answer. More likely, he 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 at hand 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.

These ideas of approximate reasoning are currently cause for 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 [1].

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 favour of the usage of expressive formalisms:

- Real and comprehensive declarative modelling should be possible. A content expert wanting to describe a domain as comprehensive and as precise 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 to reuse 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.

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 to the contrary, we argue that in particular for approximate reasoning, it is of utmost important 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.

In this paper, we thus lay the foundations for a statistical approach to evaluating approximate 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 paper refers to approximate reasoning algorithms on data which is *not* uncertain in this sense.[3]

While approximate reasoning methods for propositional and first-order logic have been proposed (see e.g. [2–10]), they are only now being applied in the context of OWL reasoning for Semantic Web technologies. Notable recent papers papers in this area are [11–18] — and to the best of our knowledge, this list should be almost exhaustive.

The paper is structured as follows. In Section 2, we will establish a mathematical framework as a foundation for approximate reasoning notions and evaluation. In Section 3 we will discuss composition of approximate reasoning algorithms from the perspective of our framework. In Section 4 we show how to instantiate our framework by means of an example. We conclude in Section 5.

[3] 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.

2 A mathematical framework for the study of approximate reasoning

In this section, we will establish a mathematical framework. By doing this, we will provide a formal basis for central notions of the field and establish guidance for lines of further research in that area.

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 conceptualisation in the following way: we allow an algorithm to – roughly spoken – 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 those 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). 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). However in most cases, it will be easy to come up with some plausible constraints for that distribution: for instance it will be very likely, that short queries will have a higher probability than long ones.

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 it occurs less often.

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. 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 cases, it might be also useful to put more constraints on the error function, one could e.g. require it to be a metric,[4] 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 $\{true, false\}$ or other nominal scales[5]), it might by 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 $\perp$ 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 $\perp$ before giving the ultimate result. $\perp$ will also be used as output value in case the algorithm does not terminate on the given input.

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

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.

So, let $\mathcal{A}$ be a set of algorithms. To every algorithm $a \in \mathcal{A}$ we assign an IO-FUNCTION $f_a : \Omega \times \mathbb{R}^+ \to X \times 2$ 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

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

[4] i.e. a distance function as used in the mathematical theory of metric spaces

[5] although also these cases can seamlessly be covered by choosing a discrete error function

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

By $f_0:\Omega\to X$ 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 $\varrho_a:\Omega\to\mathrm{I\!R}^+_\infty$ by setting

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

being the smallest time, at which the algorithm a applied to input ω indicates its termination. Note that this implies $\varrho_a(\omega)=\infty$ whenever we have $f_a^{\mathrm{term}}(\omega,t)=0$ for all $t\in\mathrm{I\!R}^+$. Algorithms, for which for all $\omega\in\Omega$ we have that $\varrho_a(\omega)<\infty$ and $f_a^{\mathrm{res}}(\omega,t)=\perp$ for all $t<\varrho_a(\omega)$ are called ONE-ANSWER ALGORITHMS: They give only one output which is not $\perp$, and are guaranteed to terminiate[6] in finite time.

Clearly, for a given time t, the expression $e(f_a^{\mathrm{res}}(\omega,t),f_0(\omega))$ 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

$$\delta:\mathcal{A}\times\mathrm{I\!R}^+\to\mathrm{I\!R}^+:\delta(a,t)=E(e(f_a^{\mathrm{res}}(\omega,t),f_0(\omega)))=\sum_{\omega\in\Omega}e(f_a^{\mathrm{res}}(\omega,t),f_0(\omega))P(\omega).$$

Note that E denotes the expected value, which is calculated by the rightmost formula.[7] 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

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

By the constraint put on the IO-function we get

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

if a terminates for every input.

[6] 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.

[7] 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.

2.1 Comparing algorithms after termination

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

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

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[8] 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

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

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:

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.*

The just given definition is very strict; a more flexible one will be given blow, when we introduce the notion that an algorithm a is *better than* an algorithm b.

2.2 Anytime behaviour

The definitions just given in Section 2.1 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.

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

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

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

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

[8] 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 2. *We say that an algorithm $a \in \mathcal{A}$ is an* ANYTIME ALGORITHM *if it realizes a defectless approximation. We say that it is a* MONOTONIC ANYTIME ALGORITHM *if is is an anytime algorithm and furthermore $\delta(a,t)$ is monotonically decreasing in t, i.e. if $\delta(a,\cdot) \searrow 0$.*

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

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

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.[9]

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 $t \geq t_{\mathrm{timeout}}$. Usually earlier results are preferred to later ones which would justify the choice of an $\bar{f}$ that is monotonically decreasing.

3 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 $\mathbf{c}$ 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 $a_{1,\ldots,n,\mathbf{c}}$ with IO-function

$$f_{a_{1,\ldots,n,\mathbf{c}}}(\omega,t) = c(a_1(\omega,t), \ldots, a_n(\omega,t)).$$

[9] 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.

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 $x_1, \dots, x_n \in X$ we have

$$c(a_1(x_1, 1), \dots, a_n(x_n, 1)) = (x, 1)$$

for some x. This is just to ensure $\varrho_{a_1,\dots,a_n,\mathbf{c}}(\omega) \leq \max\{\varrho_{a_1}, \dots, \varrho_{a_n}\}$, 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 $a_1, \dots, a_{n-1}$ are one-answer algorithms and that a_n is an (always terminating) sound and complete algorithm. Let $\mathbf{c}$ be such that

$$c(a_1(\omega, t), \dots, a_{n-1}(\omega, t), a_n(\omega, 1)) = (f^{\text{res}}_{a_n}(\omega), 1).$$

Then it is easy to see that $a_{1,\dots,n,\mathbf{c}}$ is anytime.

If we know about soundness or completeness properties of the algorithms $a_1, \dots, a_{n-1}$, then it is also possible to guarantee that $a1, \dots, n, \mathbf{c}$ 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 our inputs consist of class descriptions C over some description logic L, and outputs consist of pairs (A, B) of sets of (named) individuals. Intuitively, A contains the 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. Let $a_1, \dots, a_n$ be sound but incomplete 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. $f^{\text{res}}_a(C, \rho_a)$ – 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 $f^{\text{res}}_{a_i}(C, \rho_{a_i}) = (C_{a_i}, \mathbf{I})$ and $f^{\text{res}}_{b_j}(C, \rho_{b_j}) = (\emptyset, C_{b_j})$ 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.

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

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.

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 $a_1, \ldots, a_n, b_1, \ldots, b_m, a, \mathbf{c}$ is monotonic anytime. It is also clear that $a_1, \ldots, a_n, b_1, \ldots, b_m, a, \mathbf{c}$ is more precise than any of the a_i and b_j, at all time points.

4 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 3.

Consider the three algorithms Screech-all, Screech-none and KAON2, as discussed in [19]. We do not intend 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[10] 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

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

[10] `http://logic.aifb.uni-karlsruhe.de/wiki/Screech`

> 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:

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

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 ∞.

After this preconsiderations, we are ready to carry out some calculations estimating the quality of the considered algorithms. Figure 1 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

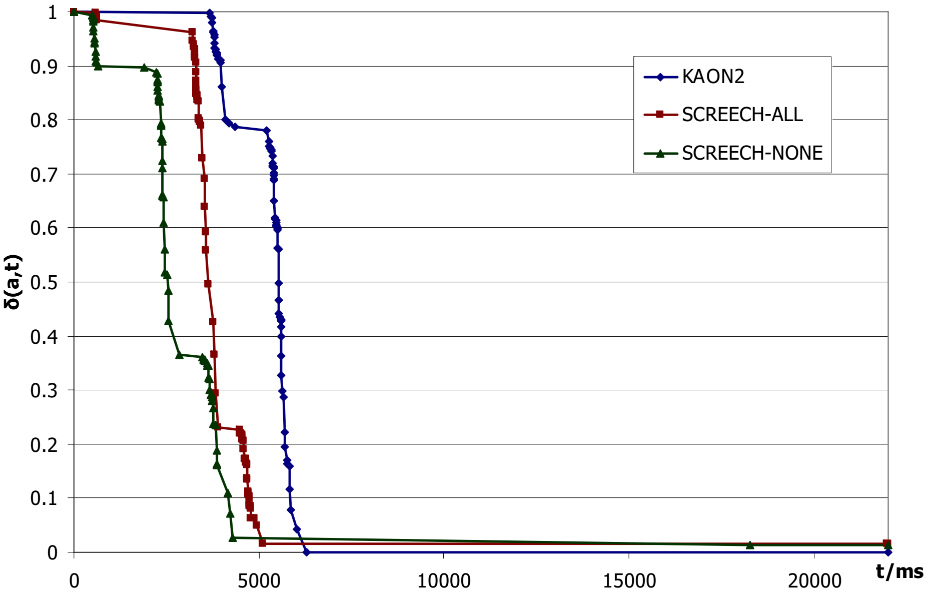

Fig. 1. Defect over time.

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 utilise a variant of the oracle given in the example from Section 3. 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 Conclusions

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 our paper, 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.

References

1. Fensel, D., Harmelen, F.V.: Unifying reasoning and search to web scale. IEEE Internet Computing **11** (2007) 94–96
2. Dowling, W.P., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. Journal of Logic Programming **1** (1984) 267–284
3. Horvitz, E.J.: Reasoning about beliefs and actions under computational resource constraints. In Kanal, L.N., Levitt, T.S., Lemmer, J.F., eds.: Uncertainty in Artificial Intelligence 3. Elsevier, Amsterdam, The Netherlands (1987) 301–324
4. Selman, B., Kautz, H.A.: Knowledge compilation using Horn approximations. In: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91). (1991) 904–909
5. Schaerf, M., Cadoli, M.: Tractable reasoning via approximation. Artificial Intelligence **74** (1995) 249–310
6. Cadoli, M., Schaerf, M.: Approximate inference in default reasoning and circumscription. Fundamenta Informaticae **23** (1995) 123–143
7. Dalal, M.: Anytime clausal reasoning. Annals of Mathematics and Artificial Intelligence **22** (1998) 297–318
8. Cadoli, M., Scarcello, F.: Semantical and computational aspects of Horn approximations. Artificial Intelligence **119** (2000)
9. van Harmelen, F., ten Teije, A.: Describing problem solving methods using anytime performance profiles. In: Proceedings of ECAI'00, Berlin (2000) 181–186
10. Groot, P., ten Teije, A., van Harmelen, F.: 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 (2004)
11. Stuckenschmidt, H., van Harmelen, F.: Approximating terminological queries. In Larsen, H., et al, eds.: Proc. of the 4th International Conference on Flexible Query Answering Systems (FQAS)'02). Advances in Soft Computing, Springer (2002)
12. Horrocks, I., Li, L., Turi, D., Bechhofer, S.: The Instance Store: DL reasoning with large numbers of individuals. In: Proceedings of the International Workshop on Description Logics, DL2004, Whistler, Canada. (2004) 31–40
13. Groot, P., Stuckenschmidt, H., Wache, H.: Approximating description logic classification for semantic web reasoning. In Gómez-Pérez, A., Euzenat, J., eds.: The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings. Volume 3532 of Lecture Notes in Computer Science., Springer (2005) 318–332
14. Groot, P., Stuckenschmidt, H., Wache, H.: Approximating description logic classification for semantic web reasoning. In Gómez-Pérez, A., Euzenat, J., eds.: The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings. Volume 3532 of Lecture Notes in Computer Science., Springer (2005)
15. Hitzler, P., Vrandecic, D.: Resolution-based approximate reasoning for OWL DL. In Gil, Y., et al., eds.: Proceedings of the 4th International Semantic Web Conference, Galway, Ireland, November 2005. Volume 3729 of Lecture Notes in Computer Science., Springer, Berlin (2005) 383–397
16. Pan, J.Z., Thomas, E.: Approximating OWL-DL ontologies. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, AAAI Press (2007) 1434–1439

17. Wache, H., Groot, P., Stuckenschmidt, H.: Scalable instance retrieval for the semantic web by approximation. In Dean, M., Guo, Y., Jun, W., Kaschek, R., Krishnaswamy, S., Pan, Z., Sheng, Q.Z., eds.: Web Information Systems Engineering - WISE 2005 Workshops, WISE 2005 International Workshops, New York, NY, USA, November 20-22, 2005, Proceedings. Volume 3807 of Lecture Notes in Computer Science., Springer (2005) 245–254
18. Stuckenschmidt, H.: Partial matchmaking using approximate subsumption. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, AAAI Press (2007) 1459–1464
19. Tserendorj, T., Rudolph, S., Krötzsch, M., Hitzler, P.: Approximate OWL reasoning with Screech. Technical report, AIFB, University of Karlsruhe (2007) Submitted to ESWC 2007.