Torture tests: a quantitative analysis for the robustness of Knowledge-Based Systems

Perry Groot[1], Frank van Harmelen[1], and Annette ten Teije[2]

[1] Division of Mathematics and Computer Science, Faculty of Sciences, Vrije Universiteit, De Boelelaan 1081A, 1081 HV Amsterdam, the Netherlands. E-mail: {perry,frankh}@cs.vu.nl.

[2] Department of Computer Science, Universiteit Utrecht, PO Box 80.089, Padualaan 14, 3508 TB Utrecht, the Netherlands. E-mail: annette@cs.uu.nl.

Abstract. The overall aim of this paper is to provide a general setting for quantitative quality measures of Knowledge-Based System behavior which is widely applicable to many Knowledge-Based Systems. We propose a general approach that we call "degradation studies": an analysis of how system output degrades as a function of degrading system input, such as incomplete or incorrect inputs. Such degradation studies avoid a number of problems that have plagued earlier attempts at defining such quality measures because they do not require a comparison between different (and often incomparable) systems, and they are entirely independent of the internal workings of the particular Knowledge-Based System at hand.
To show the feasibility of our approach, we have applied it in a specific case-study. We have taken a large and realistic vegetation-classification system, and have analyzed its behavior under various varieties of missing input. This case-study shows that degradation studies can reveal interesting and surprising properties of the system under study.

1 Motivation

When asked about the essential differences between Knowledge-Based Systems (KBSs) and "conventional software", one often hears the claim that KBSs can deal with incomplete, incorrect and uncertain knowledge and data, whereas conventional software is typically very brittle in these respects (see e.g. [Hayes-Roth, 1984] for a very early formulation of this claim). Although, nowadays researchers no longer view this distinction as either necessary or sufficient to define a KBS, it is believed that the ability of KBSs to deal with missing or invalid data is an essential dimension of KBS validation.

There has been both practical experience and theoretical analysis over many years to back up the mentioned claim. As an example of practical experience, we cite [Preece *et al.*, 1997]: in a number of verification exercises, errors were found in the knowledge-base of KBSs which were nevertheless still functioning at acceptable levels. As an example of theoretical analysis, we point to our own work [tenTeije & vanHarmelen, 1996,tenTeije & vanHarmelen, 1997] where we proved that for a large class of diagnostic systems the computed set of diagnoses degrades gracefully and predictably when either the system input (observations) or the knowledge-base degrades in quality.

However, until now, the analysis of the robustness of KBSs in the face of incomplete, incorrect or uncertain knowledge and data has been limited to such practical experience and qualitative analysis. Little or no attempt has been made at a quantitative analysis of the proclaimed robustness of KBSs. A recent special issue of a journal was dedicated to methods for Evaluating Knowledge-Based Systems [Menzies & van-Harmelen, 1999]. None of the papers in that special issue performed any quantitative analysis on the quality of Knowledge Based Systems. The editorial of this special issue lists only a hand-full of quantitative evaluation studies that have been performed over a decade or more of KBS research. In fact, one paper in that special issue [Shadbolt *et al.*, 1999] even seems to suggest that global qualitative evaluations are about as much as we can expect from KBS evaluation projects. Finally, one of the reviewers of this paper even remarked that "For a long time, the KA community has decried the lack of good evaluation metrics to measure the quality of the KA process and of the resulting knowledge bases." We consider this a serious defect in the study of KBSs, particularly since such robustness is often proclaimed as a unique characteristic of KBSs.

The central claim of this paper is that a quantitative analysis of the robustness of KBSs is both possible and useful.

To argue this claim, we present a case-study in which we perform such a quantitative analysis for a particular KBS. In section 2 we describe our approach to measuring robustness by degradation studies, and we give definitions for the basic notions involved in such degradation studies. In subsequent sections, we apply this approach in a case-study. Section 3 describes the KBS which we subjected to a degradation study and section 4 describes and analyzes the results that we obtained in this study. Section 6 summarizes the main points of the paper and looks at future steps to be taken.

2 Approach and foundational definitions

In this section we describe our approach to measuring robustness by degradation studies, and we give definitions for the basic notions involved in such degradation studies. Our aim is to define a very general set of notions that can be widely used in future degradation studies. We regard this section as a central contribution of this paper: the definitions in this section should form the basis of similar analyzes by other researchers and practitioners.

2.1 Robustness and degradation

The IEEE Standard Glossary of Software Engineering Terminology [IEEE, 1990] gives the following definition for robustness:

Informal Definition 1 (Robustness). *The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.*

In other words, robustness of a KBS is concerned with the way in which the quality of the KBS output degrades as a function of a decrease in the quality of the KBS input. This definition immediately leads to the idea of degradation studies:

Informal Definition 2 (Degradation Study). *In a* degradation study *we gradually decrease the quality of the KBS input, and measure how the KBS output quality decreases as a result.*

2.2 Output quality

Of course, we must be more precise about the rather vague notion of "quality" of the KBS input and output. Concerning the KBS output, we assume that this is always a *set* of answers. In fact, for many typical KBS tasks, this is a realistic assumption: a set of consistent classes in a classification task, a set of likely hypotheses in a diagnostic task, a set of potential designs in a configuration task, etc[i]. More explicitly stated:

Assumption 1. *For the KBSs that we consider we assume that their output can be interpreted as a discrete set of answers.*

Under this assumption, we will define two measures for KBS output quality. Let *correct (I)* be the set of all correct answers for a given input *I*, and *output (I)* be the set of actually computed answers for the input *I*.

Definition 1 (Recall). *The* recall(I) *of a KBS for a given input I is defined as:*

$$recall(I) = \frac{|correct(I) \cap output(I)|}{|correct(I)|}.$$

In other words: the recall is the fraction of correct answers that the system actually computes. It can of course happen that $correct(I) = \varnothing$ (when the system is presented with a case I for which no correct output exists, such as an inconsistent set of observations for a classification system, or an inconsistent set of requirements for a design system). In this case we define:

$$\text{if } correct(I) = \varnothing \text{ then } recall(I) = 1.$$

This reflects the intuition that the empty answer is always included in any set of answers.

Definition 2 (Precision). *The* precision(I) *of a KBS for a given input I is defined as:*

$$precision(I) = \frac{|correct(I) \cap output(I)|}{|output(I)|}.$$

In other words: the precision is the fraction of computed answers that are actually correct. In the case $output(I) = \varnothing$ (i.e. when the system returns no output), we define:

$$\text{if } output(I) = \varnothing \text{ then } precision(I) = \begin{cases} 1 & \text{if } correct(I) = \varnothing, \\ 0 & \text{otherwise.} \end{cases}$$

This reflects the intuition that the only correct answer in this case is the empty set.

[i] Of course, even for KBSs which return a single answer, this assumption still holds since we can interpret such a single answer x simply as the singleton set $\{x\}$.

We will simply write *recall* and *precision* when we mean the average of *recall*(I_j) and *precision*(I_j) over a given set of inputs $I_1, ..., I_n$.

It is a widely known fact in information retrieval that in general it is very hard to achieve both a high precision and a high recall. In general, a high recall must be paid for with a low precision. Consider for example a trivial system which always returns all candidate answers. By inspecting the definitions above, it can be easily seen that such a system would have a recall of 1, but a precision close to 0.

There are two very attractive aspects to these definitions for the "quality of output". First of all, these definitions are well known from the literature on information retrieval (e.g. [Salton & McGill, 1983]), and have proven to be useful, informative and intuitive measures in many studies in that field and elsewhere (see for instance [Fischer & Schumann, 1997] for an application of these measures to deduction-based software component retrieval). Secondly, these measures are completely general, and make no commitment to either the task or the domain of the KBS that we wish to study. Consequently, the approach proposed in this paper can be directly applied to other KBSs, even when they are very different from the one that we happen to have chosen in our own case-study.

The above definitions are in fact gradual versions of the classical notions of soundness and completeness: recall corresponds to the degree of completeness of the system, and precision corresponds to the degree of soundness of the system. These measures provide a quantitative angle on our earlier work [vanHarmelen & tenTeije, 1998] which was strictly qualitative.

2.3 Input quality

Unfortunately, the definition of input quality cannot in our view be defined in an equally generic manner. Our case-study is concerned with a classification task, and our input quality measures are directly based on this task-type, such as missing and incomplete observations. We expect that each task-type will come with its own measure for input quality.

Notice that we have already taken a step further than in [vanHarmelen & tenTeije, 1998]. In that paper we did not commit to any definition of quality on input or output, and only demanded that whatever the definition was, it should respect a partial ordering. In this paper on degradation testing, we commit to a specific definition of output quality, while leaving input quality open to be defined for each specific application.

2.4 Comparing robustness

The only notion that is still left undefined is some ordering on robustness: when do we call a system more robust or less robust than another? Unlike output quality (where we have given a single widely applicable definition) and input quality (whose definition is deliberately left open to depend on the task-type), we have not been able to determine a good answer to this question. Instead, we offer a number of competing definitions:

Definition 3 (Monotonicity). *A robust system will show a monotonically decreasing output quality as a function of deteriorating input quality.*

The motivation for this property is that a system whose output quality oscillates as a function of input quality is much less predictable than a system with monotonically decreasing output quality. This demand corresponds precisely to the usual demand on anytime algorithms that their output quality monotonically increases with increasing run-time [Dean & Boddy, 1988].

Definition 4 (Quality Value). *A system S_1 is more robust than a system S_2 for a set of inputs, if everywhere on that input set the output quality of S_1 is higher than the output quality of S_2.*

Definition 5 (Rate of Quality Change). *A system S_1 is more robust than a system S_2 for a set of inputs, if everywhere on that input set the output quality of S_1 decreases more slowly than the output quality of S_2.*

Definition 6 (Integral of Quality Value). *A system S_1 is more robust than a system S_2 for a set of inputs, if on that input set the integral of the output quality of S_1 is larger than the same integral for S_2.*

Formally speaking, definition 4 compares the output quality of two systems (and is concerned with which system produces the *best* output), while definition 5 compares the first derivative of the output quality of the systems (and is therefore concerned with which system produces the *most stable* output). Definition 6 compares the overall quality of the output quality over an entire interval even when neither system always dominates the other (as required in definition 4).

These definitions are illustrated in figure 1[ii]

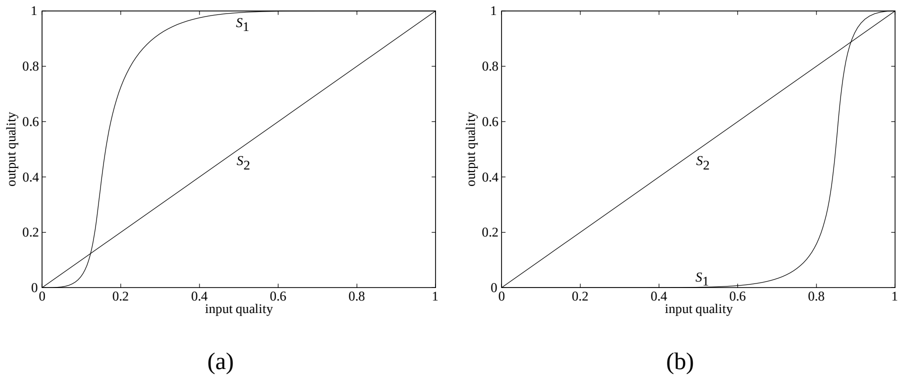


Fig. 1. Comparing robustness of systems.

In figure 1(a), system S_1 is more robust on for example the interval [0.1, 0.3] according to definition 4, since on that interval its output quality is always higher than

[ii] In our figures, we plot input quality against output quality. When speaking about “robustness”, we are interested in *decreasing* input quality, so the graphs must be read from right to left.

that of S_2. However, according to definition 5, S_2 is the more robust of the two, since its output quality decreases more gradually (reading the graph from right to left). Definition 6 allows to take a more overall perspective: it takes the size of the area under the output quality graph as measure of the overall quality. Under this definition, S_1 is more robust on the entire interval [0,1], since the value of $\int_0^1 output\ quality\ d(input\ quality)$ is larger for S_1 than for S_2. Note that the situation is rather different in figure 1(b). Although the output quality again increases from 0 to 1 over the same interval, the comparisons between S_1 and S_2 on various subintervals are very different.

At the current point in our research, we simply propose each of these definitions as reasonable, without claiming superiority of any definition in all cases. In fact, we believe that under different pragmatic circumstances, different definitions will be preferable: if steep drops in system performance are to be avoided, the second definition is preferable. If one is interested in upholding output quality as long as possible in the face of declining input, the other two definitions may be preferred.

3 Example Knowledge-Based System

An essential aspect of our proposed approach to measuring robustness through degradation experiments is that it is entirely independent of the particular problem-solving method employed by the KBS under study. All that is required is a description of the functional I/O-relation of the system[iii]. To emphasize this point, we will describe the particular KBS that we used in our case-study only in terms of its functionality, and refrain from describing the underlying problem-solving method.

For our case-study we have used a classification system for commonly occurring vegetation in Southern Germany. The plant-classification system was created with the D3 Shell-Kit which is a tool for the development of KBSs. We will not discuss this tool here but refer to a number of publications about D3 [Puppe *et al.*, 1996,Puppe *et al.*, 1994]. It is also possible to download a demo-version of the software from the URL `http://d3.informatik.uni-wuerzburg.de`.

The plant-classification system that we studied can have 40 different observables as input and has 93 different plant-names as output. The knowledge base consists of 7586 rules. Furthermore, with the system we received 150 test-cases. Each of these cases consisted of the set of observations for that case (color and shape of flowers, leafs, stem, etc.), together with the (supposedly correct) answer for these observations as given by a human expert. Around 97% could be answered correctly by the system.

The input observations can be entered in a graphical user interface, but the user is not restricted to the ordering in this interface. The observations can be entered in any order, thus the input can be seen as a *set*. This is not entirely true because some observations are dependent on other observations and will only appear when certain input-conditions are met. Because of these dependencies, the maximum number of observations that can be given for one case is 30.

The plant-classification system computes a score for every output class based on the given input. These scores result in an ordering on the output classes. All the classes with a sufficiently high score can be seen as a plausible candidate for the given input,

[iii] Sometimes referred to as “competence” in the Knowledge Engineering literature.

but plants with a higher score are seen as more plausible candidates. All the scores are computed for a given input set and these scores are adjusted incrementally when new observations are added. The user does not actually see the numeric scores of the candidates, but only an interpretation of the scores. For example, all plants with a score lower than -42 receive the status *ausgeschlossen* (i.e. excluded).

4 Results of the degradation study

In this section we present the robustness measures that we have obtained in empirical experiments with the plant-classification system. Before we discuss these results in some detail, we want to emphasize that this case-study only serves to illustrate our general approach to measure robustness through degradation testing. What's important in this section is the quantities we've decided to measure, and how we analyze them, not so much the specific results for the plant-classification system, since it only serves as a case-study to illustrate our proposal.

4.1 Which input quality measure to use?

According to the definitions from section 2 we must still decide on what to use as a measure on the input quality. In this case-study we choose the *completeness of the input* as the measure of input quality. In our classification system, completeness of the input can be directly translated as the *number of available observations*.

There are two reasons why this choice is reasonable and attractive: - **robustness:** in many practical classification settings, the input observations are not completely available. It then becomes an interesting question how robust the system functions under such incomplete input. - **Anytime behavior:** Even when all observations are present, there are practical settings where insufficient run-time is available to process all the observations: some output from the system is required before a given real-time deadline, and not all observations can be processed before this deadline. Those observations that could not be processed before the deadline can be regarded as "missing from the input".

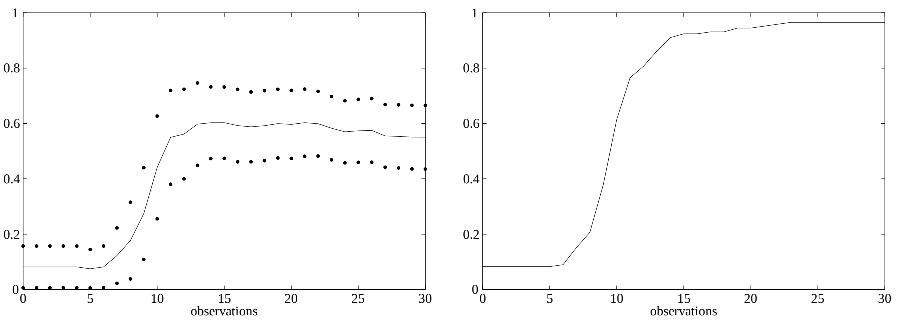
As a result of this second reason, the degradation results that we present in this section can also be seen as *anytime performance profiles* for the plant-classification system. Performance profiles are a basic tool in the study of anytime algorithms [Dean & Boddy, 1988]. They plot the output quality as a function of available run-time. Since available run-time can be interpreted as one aspect of "input quality", such performance profiles are simply a special case of our more general proposal: performance profiles only study output degradation as a function of decreased run-time, whereas our approach is applicable to any aspect of input quality that one chooses to model.

In the following we will present a number of graphs analyzing the robustness of the plant-classification system. Each of these graphs plot output-quality (measured by either recall or precision) against input-quality (measured by the number of observations that were available to the system). If one is interested in anytime behavior, these graphs can be read from left to right: "what happens when the system has time to process more and more of the inputs?". If one is interested in robustness, these graphs should be read from right to left: "what happens when the system is provided with fewer and fewer of the inputs?"

Before we discuss the results, a final remark must be made about the possible values of recall and precision in this case-study. Since for every case there is at most one correct answer (namely the name of the actual plant on which the observations were made), we have for any case I, $|correct(I)| = 1$ iff the case is in the knowledge base or $|correct(I)| = 0$ iff the case is not in the knowledge base. As a result, the only values that $recall(I)$ can assume are either 0 or 1. For the same reason, $precision(I)$ is either 0 or $\frac{1}{|output(I)|}$.

4.2 Using the input sequence from the test-cases

Now that we have established that the number of available observations will be the input-aspect that we will degrade in our studies, we have to decide in which order observations will be made available to the system. Our first choice is simply based on the order in which the observations appeared in the test-cases for the plant-classification system. Each such test-case consisted of a list of observables and their values for that case. Figure 2(a) shows how the precision of the answers from the system (as defined in definition 2) increases when longer initial-sequences of the test-cases were given to the system. The first surprise that this graph has in store for us is its monotonic growth:



(a) Precision (b) Recall

Fig. 2. Measures with test-case order.

Surprise 1. *Both average precision and average recall (see figure 2(b)) grow monotonically (or: almost monotonically in the case of precision) when adding more observations. This is somewhat*[iv] *surprising since this is not true for individual cases. The classification algorithm of the plant-classification system assigns both positive and negative scores. This means that the answer-set can both grow and shrink when adding more observations. In fact, only 58% of the test-cases has a monotonically growing answer-set.*

[iv] This result is not completely surprising because it is well known that the average of several variables can indeed show a different distribution than the individual variables.

As mentioned above, such monotonic behavior is desirable from both a robustness and from an anytime perspective, so on a case-by-case basis, the plant-classification system does not score very well on this. Surprisingly, the average-case behavior of the system is apparently much better.

A second observation to make is that (as to be expected) initially the system is not able to make any sensible guess at likely solutions (this holds up to about 6 observations). For higher number of available observations, the graph is surprising for two reasons:

Surprise 2. *After about 12 observations, adding more observations does not increase the precision. This is surprising since most cases contain as much as 19-30 observations. Figure 2(a) suggests that 12 observations is sufficient to obtain the maximally achievable precision on average.*

Surprise 3. *The region in which additional observations actually contribute to an increase in precision is surprisingly small, namely between the 6 and 12 observations. Of the 19-30 observations per case, all the real work seems to be done by this small segment of observations!*

Figure 2(b) shows similar results for the other dimension of output quality, namely the recall from definition 1.

The dotted lines in figure 2(a) indicate the variance of the precision, and this variance is rather significant. It shows that the distribution of the actual precision-values that were obtained for the different cases are actually spread rather widely around the average[v].

Figure 3 gives more insight in the distribution of the precision than the simple average from figure 2(a). Each line in this figure shows the percentage of cases that achieved a precision of at least a certain value after the given number of observations. The lowest line shows that after 12 observations, 40% of the cases have already reached the maximum precision (namely 1). Furthermore, and more surprisingly, this percentage then stops growing! This means that:

Surprise 4. *When aiming for the maximum precision of 1, there is no need to use any more than 12 observations (out of a maximum of 30!). If the maximum precision has not been reached after the first 12 observations, adding further observations will not help.*

This is actually a more precise version of surprise 2 above. There we claimed that extending beyond 12 observations was not useful *on average*. Here we see that for harder cases, a few more observations do actually help, although not more than 20 observations in total.

This is because at the other end of the scale (the top line in figure 3) , we see that the percentage of cases with a precision of at least 0.2 continues to increase during a longer interval. Apparently, harder cases (those that ultimately achieve a lower precision) benefit more from additional observations than easy cases (those that achieve precision 1). Nevertheless, even there we see that no increase is gained after about 20 observations:

[v] No variance was plotted for the recall since, as explained above, in our application the recall is either 0 or 1. Because of this, the variance for recall is not a meaningful notion.

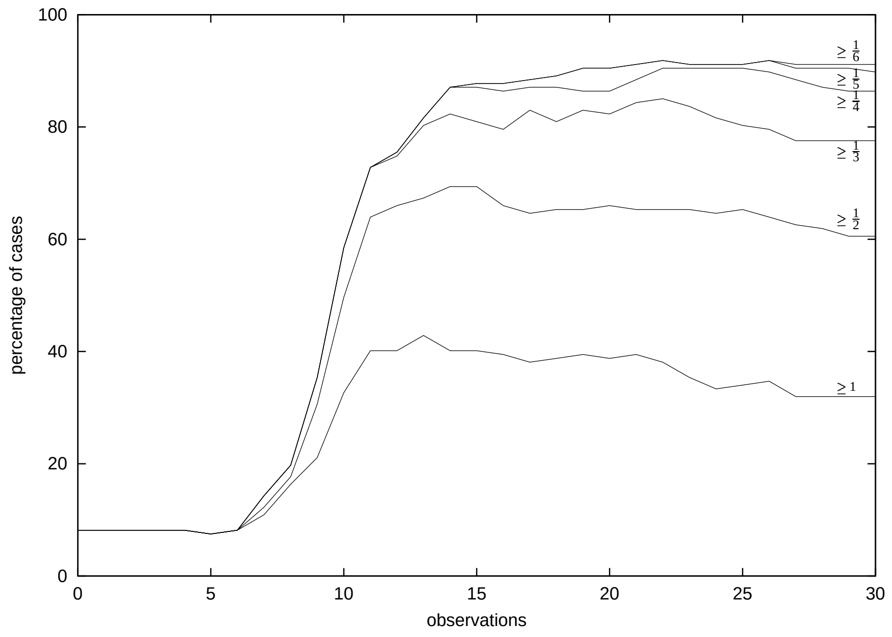


Fig. 3. Precision with multiple levels.

Surprise 5. *Whatever the final precision that is ultimately obtained by the system, this level of precision is already obtained after at most 20 observations. It seems that asking for any more then 20 observations will not improve the output quality any further. This is surprising since many cases (in fact 98% of the test set) contain more than 20 observations.*

Looking at the initial segment of observations, we see another surprise: although we may expect that a low number of observations leads to a low average precision, it is surprising that the lines for the different precision-levels all *coincide* until the 6th observation:

Surprise 6. *No increase in precision can be gained from the first 6 observations.*

This means that in an anytime setting, interrupting the system before the 6th observation is completely useless, since no increase in precision will have been obtained yet.

Figure 3 is particularly interesting from an anytime perspective: it tells us for each partially processed input what the chance is that the system has already obtained a certain precision in its output: for instance, after having fed the system 10 observations, there is a 30% chance that it has already obtained the maximum precision of 1, a 45% chance that it has already obtained a precision of at least 0.5, and a 60% chance that it has already obtained a precision of at least 0.3.[vi] This information can be used by

[vi] Note that these chances are cumulative, which is why they add up to more than 100%.

the user to determine if it is useful to continue feeding the system more input, or if a sufficiently high precision has already been obtained for the purposes of the user, so that processing (and acquiring potentially expensive observations) can be stopped. Our graph (when interpreted as a performance profile) contains much more information than the usual performance profiles presented in the literature (e.g. [Zilberstein, 1996]). These graphs typically give only a *single* expected value for the output quality at any point in time (compare our figure 2(a)), whereas we give a probability distribution of the expected output value, which is much more informative.

Note that the ideas behind figure 3 can in principle also be applied to the recall. We omitted this because in our case -study the recall is either 0 or 1, thus the resulting figure would be the same as figure 2(b).

4.3 Early conclusions on robustness

Since we are only studying a single system, we cannot apply the definitions from section 2.4, which only speak about one system being more or less robust than another. However, what is clear from the analysis until now is that the robustness of the plant-classification system is certainly not very uniform across the distribution of input quality. While degrading input quality (i.e. reading the previous graphs from right to left), the system at first appears extremely stable against missing observations: no quality loss occurs at all. This holds until we are left with somewhere between 12-15 observations (depending on the difficulty of the case). At that point, the robustness of the system is very low, and the input quality drops dramatically.

Is this desirable behavior or not? Would a more uniform behavior (e.g. a straight line connects bottom-left and top-right of figure 2(a)) be more attractive? In our view, this question cannot be answered in general, but depends on the pragmatics of the system in use. The “straight line” profile is on the one hand more attractive, because it avoids the dramatic drop in quality seen in the figures above (definition 5 from section 2.4); on the other hand, it would start loosing output quality straight away, while the profiles discussed above are all remarkably resistant to quality loss during early phases of input degradation (definition 4 from section 2.4).

4.4 Exploring other input sequences

In all the profiles above, we have degraded the input by removing observations in the order in which they were listed in each test-case. Figure 4 shows what happens if the input is degraded by removing observations in a different order.

For reference, the dotted line shows the recall-profile from figure 2(b). The left-most line shows the theoretically optimal average-recall profile: at each step in each case, we computed which next observation would contribute maximally to an increase in recall. Of course, this cannot be done in practice, since which observation will contribute most in general depends on the observation-value that is obtained, so this computation can only be done theoretically for test-cases where all observations are already present. The value of the left-most line is therefore only to show what would be the theoretically fastest increase in recall by the system with the fewest possible observations. The right-most line in figure 4 does the same, but this time for the theoretically slowest

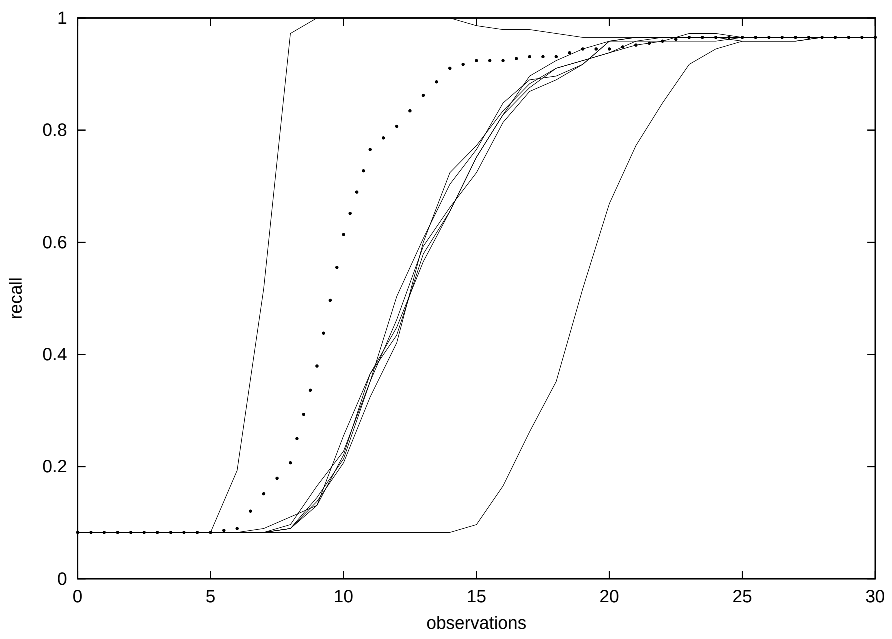


Fig. 4. Recall with various orderings.

average-recall profile. Every other possible recall profile must lie between these two lines (as is indeed the case with the earlier observed profile based on the test-case sequence). Finally, figure 4 shows a narrow bundle of recall profiles. Each of these profiles corresponds to a randomly generated order of the observations.

The dotted line in this figure shows the profile based on the observation order obtained from the test-cases (as originally plotted in figure 2(b)). We can now see that this observation order actually scores rather well when compared with the random sequences:

Surprise 7. *The degradation sequence taken from the test-cases is surprisingly effective in obtaining a high recall after only a few observations. In fact, it is much closer to the theoretically optimal sequence than the randomly generated (information-free) sequences.*

Currently, we have no good explanation for this phenomenon. It is possible that the order of the input observables in the test-cases is influenced by the order in the graphical user-interface. We will look more closely at this ordering in section 4.6.

4.5 Further conclusions on robustness

The variation in the curves from figure 4 shows that the plant-classification is very sensitive to the specific order in which the observations are presented to the system. In other words: when the same set of observations are presented to the system in a different order, the behavior of the system may change dramatically.

This type of robustness is not covered by our robustness definitions in section 2.4. The definitions there are all concerned with comparing the behavior of different systems on the same degrading input. The phenomenon observed in figure 4 concerns the behavior of a single system on different ways of degrading the input. We leave it for further research how to include this type of robustness in our approach.

4.6 Which sequence does the system actually use?

The original plant-classification system has been designed in such a way that all observations can be entered at any time during the dialogue. The system does not enforce a particular order among the observations to be entered as input. Nevertheless, the user-interface of the system does suggest a particular input sequence, namely the order in which the observations occur on the input form. Although the user can enter observations anywhere from the input form, the top-to-bottom sequence of this form is suggestive.

The solid line in figure 5 shows the increase in recall for this user-interface sequence. The dotted lines show the theoretically fastest, the theoretically slowest gains in recall, and the gains from random observation orderings and the test-case ordering as well (all copied from figure 4). As can be observed in this figure, the user-interface sequence performs rather well when interpreted as an anytime performance profile (and certainly much better than a random ordering). This suggests that the user-interface has to some extent been designed with this behavior in mind. Although this behavior was not the reason for the plant-classification system, it would be interesting to see how a dedicated dialogue strategy can save time.

Actually, the real reason behind the ordering of the user-interface was the intended use of the system. This means that we have to place our results in this context. The ordering of the user-interface is based on the way people describe plants. First the system asks about the flower, which is the most specific part of the plant and asks about the leaves and stem afterwards. This specificity ordering may be the reason for the better curve in figure 5 when we compare it with a random ordering.

Furthermore, we like to note that the most optimal input order w.r.t. the recall or precision, may not be realizable in practice because it can decrease other factors like for example the user-friendliness of the user-interface. Some trade off will usually have to be made.

5 Future Work

Although the results above already yield interesting insights in the behavior of a realistic KBS, many other aspects could still be uncovered using further degradation studies. We discuss some of these extensions in this section:

The informal definition for robustness that we used as a starting point in section 2 has not been carried through entirely in the paper. Functioning correctly in the presence of invalid inputs has not been evaluated in the case study and should be included in future research.

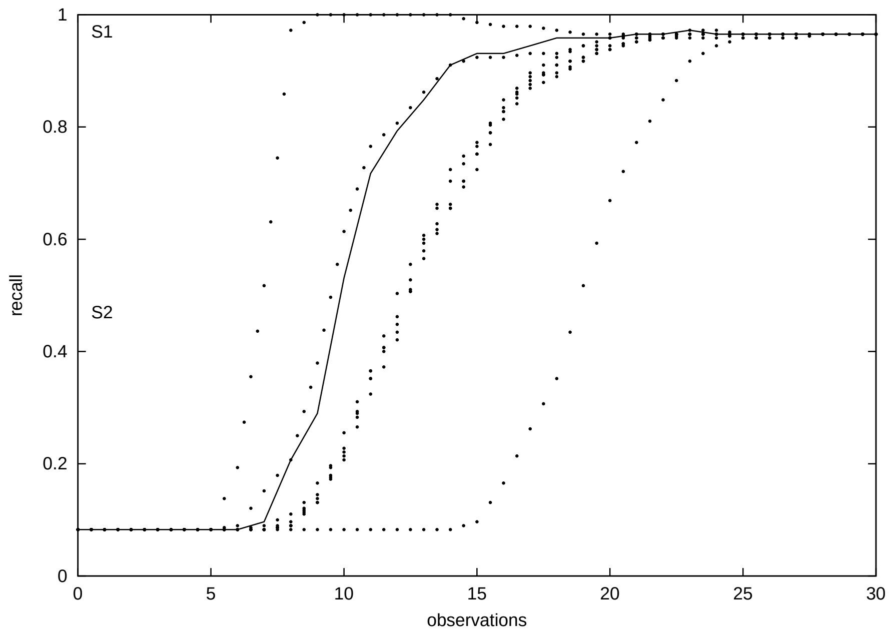


Fig. 5. Recall with user-interface ordering.

[Zilberstein & Russell, 1995] suggests a category of measures on output quality which is not yet covered by the recall and precision that we have used in the above, namely "specificity of a solution". This is intended to represent the degree of detail in the systems' answer. An example would be a system which can compute names of ever finer grained plant-families instead of only individual species (as above). This property was irrelevant in our case-study, since the plant-classification system only deals with a flat list of candidates, not with a hierarchically organized space of candidates. We have therefore ignored this potential third dimension of KBS output quality, but we expect that good measures can be devised for this just as well as for the other two dimensions which we did handle.

In this paper, we have studied the consequences of degrading the *input* quality of a KBS. It would be equally if not more interesting to study the effects of degradations of the knowledge-base itself. After all, the knowledge-base is most likely incomplete and often partially incorrect. We expect that the same approach as taken in this paper will apply to such a knowledge-base degradation study, and we intend to perform such a study in the near future.

The output quality measures we used (precision and recall) are geared towards systems with a discrete output (a set of answers). Some KBS applications return real-valued answers (e.g. ratings). We must study how these systems can also be subjected to degradation studies using acceptable measures. In fact, the plant-classification system not only returns a set of candidates, but indicates a numeric score for each candidate. Our current output-measures completely ignore this score. A further step would be to also include this score in the quality measures.

As mentioned in section 4.2, figure 3 can be interpreted as a prediction for the expected quality of the output after a given number of observations. In effect, figure 3 is the result of *learning the anytime performance profile* through the test-cases. As with any learning task, we can apply cross-validation to the set of test-cases [Cohen, 1995]: use a subset of the test-cases to “learn” profiles as in figure 3, and use the remaining cases to check the accuracy of the predicted performance levels.

Our measures for output quality (recall and precision) can only be computed for cases where the correct answer is actually known. This is not as obvious as it may sound. In many applications (e.g. computing the best solution to a design problem) the correct (i.e. best) answer is not known to any human expert. In such cases, one must either resort to known approximations of the correct answer, or fundamentally different quality measures must be defined.

Our proposed definitions for output quality (precision and recall) focus on the correctness of the answers computed by a system. Of course, there are many more aspects to the “quality” of a KBS, such as the quality of its explanation, its computational efficiency, its interaction with its environment (be it users or other systems), etc. It is an open issue to us whether the same “degradation study” approach can be taken to quantifying any of these other aspects of the systems quality.

6 Conclusions

In this paper, we have argued for the need for *quantitative analysis* of the quality of KBSs. In particular, we have shown how *robust behavior* in the light of incomplete system-input is amenable to such quantitative analysis. Our quantitative analysis is based on the idea of *degradation tests*: analyze how the quality of the output degrades as a function of degrading input. We have proposed a set of *general definitions* which are general enough that they can be used in similar degradation experiments by others, even if the systems concerned are of a very different nature than the one in our case-study. The proposed approach of measuring robustness via degradation experiments is *entirely independent of the problem-solving method* used internally by the KBS under study. All that is required is a functional description of the I/O-relation of the KBS. We have shown the practicality of our approach by applying it to a particular *case-study*. This yielded a number of surprising insights into the behavior of the system under study.

The ultimate suggestion that follows from this work is that any KBS should upon delivery come accompanied with a set of degradation statistics such as discussed in this paper as a quantitative way of measuring interesting and important aspects of the systems quality. This would contribute to a more empirical and quantitative analysis of AI systems in general and of KBSs in particular, very much in the spirit of [Cohen, 1995].

7 Acknowledgements

We like to thank Frank Puppe for the help he has given us. This paper was inspired by a discussion with him during the Banff’98 KAW workshop. He gave us access to the plant-classification software and explained in detail its working. He was also always available to answer any question by e-mail.

References

[Cohen, 1995] P. R. Cohen. *Empirical methods for artificial intelligence*. MIT Press, 1995.

[Dean & Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In AAAI–88, pages 49–54, 1988.

[Fischer & Schumann, 1997] B. Fischer and J. Schumann. NORA/HAMMR: Making deduction-based software component retrieval practical. In *Automated Software Engineering (ASE)'97*, pages 246–254. IEEE, 1997.

[Hayes-Roth, 1984] F. Hayes-Roth. Knowledge-based expert systems — the state of the art in the US. In J. Fox, editor, *Expert Systems: state of the art report*. Pergamon Infotech, Oxford, 1984.

[IEEE, 1990] IEEE. IEEE standard glossary of software engineering terminology, 1990. IEEE Standard 610.12-1990, ISBN 1-55937-067-X.

[Menzies & vanHarmelen, 1999] Tim Menzies and Frank van Harmelen. Evaluating Knowledge-Engineering Techniques. *International Journal of Human-Computer Studies*, 51(4):715–727, October 1999.

[Preece *et al.*, 1997] A. Preece, S. Talbot, and L. Vignollet. Evaluation of verification tools for knowledge-based systems. *International Journal of Human-Computer Studies*, 47:629–658, 1997.

[Puppe *et al.*, 1994] F. Puppe, K. Poeck, U. Gappa, S. Bamberger, and K. Goos. Wiederverwendbare Bausteine für eine konfigurierbare Diagnostik-shell. *Künstliche Intelligenz*, 94(2):13–18, 1994.

[Puppe *et al.*, 1996] F. Puppe, U. gappa, K. Poeck, and S. Bamberger. *Wissensbasierte Diagnose- und Informationssysteme*. Springer-Verlag, Juli 1996.

[Salton & McGill, 1983] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[Shadbolt *et al.*, 1999] Nigel Shadbolt, Kieron O'Hara, and Louise Crow. The experimental evaluation of knowledge acquisition techniques and methods: history, problems and new directions. *International Journal of Human-Computer Studies*, 51(4):729–755, October 1999.

[tenTeije & vanHarmelen, 1996] A. ten Teije and F. van Harmelen. Computing approximate diagnoses by using approximate entailment. In KR'96, pages 265–256, 1996.

[tenTeije & vanHarmelen, 1997] A. ten Teije and F. van Harmelen. Exploiting domain knowledge for approximate diagnosis. In M. Pollack, editor, IJCAI'97, pages 454–459, Nagoya, Japan, August 1997.

[vanHarmelen & tenTeije, 1998] F. van Harmelen and A. ten Teije. Characterising approximate problem-solving by partial pre- and postconditions. In ECAI'98, pages 78–82, 1998.

[Zilberstein & Russell, 1995] S. Zilberstein and S. J. Russell. Approximate Reasoning Using Anytime Algorithms. In S. Natarajan, editor, *Imprecise and Approximate Computation*. Kluwer Academic Publishers, 1995.

[Zilberstein, 1996] S. Zilberstein. Using anytime algorithms in intelligent systems. *Artificial Intelligence Magazine*, fall:73–83, 1996.