BVAL: Probabilistic Knowledge-Base Validation

Eugene Santos Jr.[1] and Howard T. Gleason and Sheila B. Banks
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-7765
esantos@afit.af.mil

Abstract

Our work develops a new methodology and tool for the validation of probabilistic knowledge bases throughout their lifecycle. The methodology minimizes user interaction by automatically modifying incorrect knowledge; only the occurrence of incomplete knowledge involves interaction. These gains are realized by combining and modifying techniques borrowed from rule-based and artificial neural network validation strategies. The presented methodology is demonstrated through BVAL, which is designed for a new knowledge representation, the Bayesian Knowledge Base. This knowledge representation accommodates incomplete knowledge while remaining firmly grounded in probability theory.

Introduction

The development and use of knowledge-based systems (KBSs) has steadily increased since their inception, and seems likely to continue in this trend (Giarratano & Riley 1994)(Gonzalez & Dankel 1993). Any system, including a KBS, is of little use if it does not function properly. While many components of a KBS contribute to its overall "functional correctness" (user interface, inference engine, knowledge base), this research is concerned with the *correctness of the knowledge base*, not the entire knowledge-based system. Specifically, the critical factor focused on here is ensuring that the knowledge base provides an acceptable answer to every query. In the medical domain, this is analogous to ensuring the doctor provides the correct diagnosis, without regard for bedside manner, location of office, hourly charges, etc. This process is generally termed the "verification and validation (V&V) of the knowledge base." Verification can be thought of as ensuring the knowledge is collected and structured properly, while validation ensures the knowledge produces correct results.

Historically, the development and application of knowledge-based systems has been published before any V&V results on the same system (O'Keefe, Balci, & Smith 1987; Buchanan & Shortliffe 1984), and the trend continues today. Furthermore, while KBSs using probabilistic knowledge representations are popular, little literature exists regarding the validation or verification of those knowledge bases. It is this missing piece of information that our work addresses.

[1]This research was supported in part by AFOSR Project #940006.

Validation vs. Verification of Knowledge

The general process of testing, evaluating, and correcting any software is termed "verification and validation" (V&V). As they relate to knowledge bases, verification is the process of ensuring the knowledge base was built right, while the validation process ensures the right knowledge base was built (O'Keefe, Balci, & Smith 1987).

Verification is concerned with comparing the knowledge base against its specifications (Gonzalez & Dankel 1993; Guida & Mauri 1993). This process ensures that the transfer of knowledge from human to machine-understandable form, does not violate any constraints of the knowledge representation. In practice, the knowledge representation may not have any intrinsic constraints; however, the inferencing mechanism that reasons over this knowledge representation will probably have constraints. These inferencing constraints are usually mapped onto the knowledge representation to ensure that a verified knowledge base can be inferenced over without causing an error.

Validation, on the other hand, is concerned with ensuring that the knowledge in the knowledge-base, when inferenced over, provides acceptable responses. This "sine qua non" of knowledge-base evaluation can be defined as the comparison of the knowledge base against its requirements (Guida & Mauri 1993). Both processes include the testing for and correction of their specific types of errors.

Rule-Based V&V Approaches

Approaches to the verification and validation of rule-based knowledge bases are numerous (Gupta 1991; Zlatareva 1994; Nazareth 1993; Botten, Kusiak, & Raz 1989; Yager & Larsen 1991). While many of these sources claim to provide methods and techniques for verification and validation, nearly all of them are solely concerned with verification. This preponderance of verification literature centers around a standard list of errors, developed through the years, against which rule-bases are to be checked. This list generally includes (Gonzalez & Dankel 1993):

1. redundant rules
2. conflicting rules
3. subsumed rules
4. circular rules
5. unnecessary IF conditions
6. dead-end rules
7. missing rules
8. unreachable rules

In this list, the top five items pertain to the *consistency* of the rule-base and the last three to its *completeness.* Generally, "rule-based verification" has been transformed into "consistency and completeness" (Gupta 1991; Buchanan & Shortliffe 1984; Gonzalez & Dankel 1993).

The historical dominance of rule-based systems has extended this "verification = completeness + consistency" concept to knowledge-based systems in general. The reason this extension has proceeded unchecked is due to the lack of specifications surrounding knowledge-base development. As mentioned previously, the specifications of a knowledge-base are actually the constraints of the inferencing method. It follows, then, that the main thrust in knowledge-base verification for all KBSs has actually evolved from the constraints of rule-base inferencing. In other words, the push towards "completeness and consistency" as the goal of verification is an artifact of rule-based inferencing mechanisms, and not necessarily a requirement for all knowledge-base verifications. Any verification of a knowledge base with an inferencing mechanism able to accommodate incompleteness renders the "completeness" constraint moot.

Turning to validation, many authors recognize the need to validate the knowledge base (Buchanan & Shortliffe 1984; Gupta 1991); however, published results of actual validations are few. The three validations that MYCIN underwent are frequently cited as benchmarks in the evolution of knowledge-based *system* validation (Buchanan & Shortliffe 1984); however, *knowledge-base* validation lacks historical benchmarks save TEIRESIAS (Davis 1979). The act of "knowledge-base debugging" (Buchanan & Shortliffe 1984) is accomplished on every KBS, but for some reason rarely defined or detailed. Viewed from the definition mentioned above, knowledge-base validation has historically lacked requirements (Gupta 1991). Consequently, the techniques of knowledge-base validation have been left open for interpretation.

Neural Networks and Validation

As knowledge-based systems, neural networks (Hecht-Nielsen 1991) have a unique approach to validation. The neural net itself, metaphorically the knowledge-base, is validated through training of the net (which is akin to knowledge acquisition) and amounts to a continuous validation process. During this process, the net is presented with a set of inputs and a set of desired outputs. The input is then propagated through the net and its output is compared with the desired output. Any error in output, even if the solution was correct, is propagated backwards through the network adjusting the weights in the direction of the desired output. After each pass of the training data through the network, an accuracy percentage is calculated by passing all the training data through and determining what percentage produces the desired output. This entire process is repeated many times until the overall accuracy of the network is as high as possible. At the successful conclusion of this training, the neural net is considered validated for its training set.

Knowledge Representation

One of the difficulties traditional knowledge representations have is handling uncertainty; something humans accomplish regularly. For example, if you were told, "When it rains, the sidewalk gets wet" you would understand the meaning. Similarly, a KBS utilizing IF-THEN rules could be given: "IF raining = true THEN sidewalk-surface = wet". The uncertainty problem occurs when it is raining, but the sidewalk is not wet. A human may observe this apparent contradiction and remark "There's an exception to every rule." However, getting a KBS to accommodate this situation is not trivial. The IF-THEN rule could be modified to include exceptions, but at what point are enough exceptions listed? The main problem is that the original statement "When it rains, the sidewalk gets wet," is not completely true. It is true under *most* conditions, but not all. A knowledge representation that can incorporate uncertainty and accommodate knowledge from a variety of domains is essential in making a Knowledge-Based System that is both general and useful.

Development of the earliest systems showed that

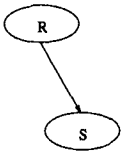

FIGURE 1. **Bayesian Network.** The Bayesian Network corresponding to the rule: "IF raining = true THEN sidewalk-surface = wet with probability of 0.9". Here "R" and "S" correspond to "raining" and "sidewalk-surface" respectively.

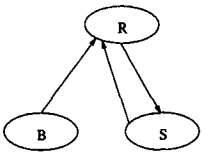

FIGURE 2. **Improper Bayesian Network.** The Bayesian Network from Figure 1 with this rule added: "IF sidewalk-surface = dry AND building-shelters-sidewalk = false THEN raining = false with a probability of 0.8".

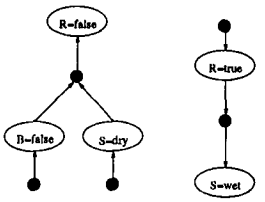

FIGURE 3. **Bayesian Knowledge Base.** The Bayesian Knowledge Base representing the two rules from Figure 2. Notice that the BKB can accommodate RV level cyclicity–something a Bayesian Network cannot.

accommodating uncertainty was a must for making a KBS that could function well in diverse domains (Buchanan & Shortliffe 1984). Currently many schemes of accounting for uncertainty in a knowledge representation exist, e.g. probabilities, certainty factors, Dempster-Shafer theory, and fuzzy logic to name a few. While some of these methods have been demonstrated in specific domains, probability theory, with its well-established mathematical foundation, provides sound and consistent knowledge representations for many domains (Pearl 1991; Santos & Santos 1996).

One of the more popular knowledge representations accommodating uncertainty is the Bayesian Network (Pearl 1991). This knowledge representation models the probabilistic dependencies between discrete objects, termed random variables (RVs). Resuming the above example, and adding probabilities to modify the rule yields: "IF raining = true THEN sidewalk-surface = wet with a probability of 0.9."[2] The more general approach taken by Bayesian Networks eliminates the IF-THEN structure, and associates "raining = true", "sidewalk-surface = wet", and "0.9." Bayesian probabilities (subjective probabilities) allow this type of association to be made. To illustrate: $P(A = a \mid B = b) = p$ means p is the probability that $A = a$ given that $B = b$. The example now becomes:

$$P(\text{sidewalk-surface} = \text{wet} \mid \text{raining} = \text{true}) = 0.9.$$

Graphically, this relationship is shown in Figure 1.

There are, however, restrictions on the arrangement of information in a Bayesian network due to theoretical limitations (Santos & Santos 1996). Continuing the example, the following rule needs to be added to the knowledge base: "IF sidewalk-surface = dry AND building-shelters-sidewalk = false THEN raining = false with a probability of 0.8", see Figure 2.

Such "circular dependencies" are fairly common to humans, however classic Bayesian networks prohibit them; limiting the flexibility and versatility of the knowledge representation. Further limiting usefulness, an exhaustive list of RV-states, as well as every dependent and prior probability, must typically be supplied to the Bayesian network before inferencing is possible. Often, these numerous (combinatoric) probabilities are not known, and do not make any sense to a human; defining them becomes a major stumbling block in knowledge acquisition.

These limitations have been overcome by the Bayesian Knowledge Base (BKB) (Santos & Santos 1996). The BKB is a generalization of the classic Bayesian network capable of incorporating more detailed probabilistic dependencies and incomplete knowledge of RV-states and dependencies. The BKB subsumes Bayesian networks, and remains firmly rooted in probability theory.

The two key differences between Bayesian Networks and BKBs are: knowledge granularity, and knowledge completeness. The BKB records probabilistic dependencies at the instance (state) level, instead of the RV level; i.e. the smallest piece of knowledge in a BKB is an RV-instance (raining = true), as opposed to the Bayesian Networks' RV (raining). Figure 3 shows the BKB needed to represent the two rules present in Figure 2.

A BKB is a bipartite directed graph composed of two

[2] For clarification, here the RV's are "raining" and "sidewalk-surface".

distinct types of nodes. The lettered ovals, termed "instantiation nodes" or "I-nodes", represent specific RV-instances. The filled circles, "support nodes" or "S-nodes", each represent one probabilistic dependency relationship, and contain the probability associated with that relationship. The BKB has no requirement for exhaustive RV-state lists or complete probability specifications. This allows it to accommodate incomplete as well as uncertain knowledge.

Knowledge Acquisition and Consistency

Any knowledge representation is of little use without a means of knowledge acquisition. The MACK tool presented in (Santos & Banks 1996) is the knowledge acquisition device for the Bayesian Knowledge Base. MACK supports an incremental knowledge acquisition philosophy, allowing the knowledge engineer freedom in constructing the knowledge base. This tool uses linguistic parameters to help overcome the typical difficulties in defining exact probabilities for a knowledge base. Additionally, it provides a mechanism for handling temporal information, and guarantees knowledge base consistency.

This consistency is actually the verification of the knowledge base. MACK compares the structure of the knowledge base against a set of constraints (specifications) that ensure the inferencing method is able to perform correctly. BVAL guarantees that the knowledge base will remain verified during its operation.

Inferencing

The purpose of any Knowledge-Based System is to provide information (diagnosis, advice, analysis, etc.) to a user. The process of deriving this "information", termed inferencing, is of central importance to any KBS.

Belief Revision (Pearl 1991) finds the "most probable explanation" (MPE) for the given evidence based on the optimal joint probability of the random variables. This is actually the most likely state of all random variables in the knowledge base; or more informally, the most likely state of the world. This type of inferencing is most useful in explanatory/diagnostic domains where the exact probability of any particular element is not as useful as the inclusion of an element. For example, in a medical domain (diagnostic) a patient may have chest pain (evidence). Inferencing with Belief Revision will result in every RV being assigned a state (instantiation). One of the RV-states in the MPE may be "congestive-heart-disease = true". The presence of this element is of more value to the user of the system and/or the patient, than is the exact probability that the patient has congestive heart disease (Belief Updating). In fact, the probability of having a given disease may be fairly small on an absolute scale; however, the fact that the disease is a part of the "most probable state of the patient" is significant.

While this inferencing method seems to provide the desired information in its applicable domains, a problem occurs when it is used on an incomplete knowledge base (BKB). The problem has two main roots: the "type" of information in the knowledge base, and the "globalness" of the inferencing solution. While these root problems seem disparate, they actually combine to address the broader topic of relevancy.

"Type" of information is best described by way of example. Remaining in the medical domain, the knowledge acquisition process will probably acquire only "positive" information; i.e. information that can be used to conclude the presence of some disease. "IF chest-pain = true THEN congestive-heart-disease = true" (positive information) is more likely to appear in the knowledge base than an exhaustive listing of exactly what will cause "congestive-heart-disease = false". It seems reasonable that the knowledge engineer, when given the freedom to supply incomplete information, will supply only the "important" or relevant information. From here, it is easy to see how the BKB would contain many "disease-X = true" instances, and few "disease-X = false" instances.

As mentioned above, Belief Revision results in every RV being assigned its most likely state with respect to the evidence. This "global" assignment of every RV will result in the most likely assignment for all "disease-X" RVs. However, since many of those RVs have only one state in the knowledge base, their state assignment is a foregone conclusion. What this means to the user is every disease with only one state, or all positive states, will be present in every solution; e.g. no matter the evidence, the patient's most likely state will include (say) flu, cold, pneumonia, meningitis, ulcer, congestive heart disease, etc..

A possible solution to this problem is to make a single "disease" RV, that has many states corresponding to flu, cold, etc.. This method, however, has problems of its own. It cannot accommodate positive and negative information for a single disease, it cannot account for one disease being a symptom of another disease (e.g. pneumonia is a symptom of AIDS), and it only allows one disease to be present in any solution (since a random variable cannot be in two states simultaneously). Each of these problems is as unacceptable as the original problem of having many diseases every time.

The ideal solution is to inference over (assign states to) only those random variables that are relevant to

the evidence.

Solution vs. Answer

The previous section mentioned, but never defined, the "solution" that results from inferencing. The solution is the set of elements found to compose the most likely (partial) state of the world. This set forms a subgraph of the knowledge base and usually has paths that include the evidence. This solution provides important information for validation (as we shall see) but presents a problem for providing an answer to the user.

To clarify, "solution" refers to the output of the inferencing mechanism, while "answer" refers to that portion of the solution presented to the user. The need for this distinction is apparent when dealing with a Belief Revision solution. Recall that a Belief Revision solution contains the state (possibly assumed) of every RV in the knowledge base. Returning to the medical example and bypassing the problems discussed previously, the evidence is "fever = true", and the solution now contains "cold = true", "shivering = true", "coughing = true", "sore-throat = true", etc. What is the *answer*?

One approach to this "interpretation" problem is to not interpret the solution at all but just display the entire solution to the user. It is easy to see, however, that such an approach quickly degrades in usefulness as the size of the knowledge base increases. Forcing the user to examine a large number of RV-states, looking for the one (few) that provide useful information is not effective. The real crux of this problem is somehow deciding what element(s) of the knowledge base are relevant to the solution given the evidence. The following approach attempts to address this problem by extending the knowledge acquisition process and adding functionality to the user interface.

This approach involves having the expert/knowledge engineer provide additional information during knowledge acquisition. This "additional information" will be the "type" or "category" of the random variable. For example, the RV "fever" might be type "symptom", the RV "cold" might be type "disease", and the RV "pneumonia" might be type "symptom" and type "disease". The exact type label(s) should be determined by the expert/knowledge engineer, since they are the source of information contained in the knowledge base.

Now that this additional information is in the knowledge base, it must provide some functionality. The user interface can utilize this information to filter the solution as reported by the inferencing mechanism. For example, if "cold = true", "shivering = true", "coughing = true", and "sore-throat = true" compose the solution, and "cold" is of type "disease" while the others are all of type "symptom", the solution can be filtered for "disease" RVs resulting in the *answer* "cold = true".

In deciding what types to report, the expert/knowledge engineer could provide a list of "reportable types" as a default filter for each query based on the purpose of the knowledge base; e.g. all RVs of type "disease" are included in the answer by default. The user may also specify types to include or exclude from the answer, giving them the flexibility to tailor the output without changing the knowledge.

Bayesian Knowledge Base Validation

Ideally, the process that ensures an acceptable solution is provided for every query would be completely automated. However, since BKBs incorporate incomplete knowledge, this is not possible. Incomplete knowledge produces errors that can only be corrected by modifying the amount of knowledge in the knowledge base. Automatically performing this correction requires an awareness of the knowledge that is either missing or extraneous. Having an awareness of what knowledge is needed to automatically correct an incomplete-knowledge error, implies that the system already contains the knowledge needed to fix the error. Completing this circular argument, if the system already knows what knowledge is missing or extraneous, why would an incomplete-knowledge error ever occur? Clearly, the allowance of incomplete knowledge requires the user to interact with the knowledge base during validation. Keeping the ideal in mind, the goal of BVAL is to minimize the interaction required.

BVAL's approach to validating Bayesian Knowledge Bases achieves this goal by combining both rule-based and neural net validation techniques. The rule-based approach emphasizes completeness and correctness while only providing the user with assistance in locating the incomplete/incorrect knowledge. The neural net approach assumes its knowledge is complete and automatically reinforces correct knowledge. Since BKBs will have incomplete and incorrect knowledge, BVAL combines these techniques to provide automatic correction whenever possible and user assistance otherwise.

Validation Issues

Before detailing BVAL's methodology, it is important to understand its overall approach in general terms. The PESKI system which contains BVAL will be released as a KBS *shell*, a fully functional system without any knowledge. Since each user will need to validate their individual knowledge base, BVAL was developed for inclusion in the system shell. Because of this ar-

rangement, BVAL is designed to directly compare the knowledge base against its requirements. The requirements of the knowledge base are a set of test cases, where each test case has evidence and an expected answer. This requirements set is similar to the training/testing sets of neural networks. BVAL is designed to be useful over, and an integral part of, the whole knowledge base lifecycle.

Four significant validation issues were defined by (O'Keefe, Balci, & Smith 1987) and detailed in (Giarratano & Riley 1994): What to Validate, When to Validate, Which Methodology and What Criteria. We now consider how they pertain to BVAL.

What to validate: Since validation is concerned with ensuring an acceptable solution is provided for every query, it seems obvious that the final result, the solution, is being validated. The intermediate results, while a part of the solution, are immaterial if the solution is correct. If an intermediate result is to be included in the solution, then it should be part of the expected answer for that test case. BVAL's methodology will then ensure that this intermediate result is a portion of the solution.

When to validate: BVAL is designed to be used at all stages of the knowledge base lifecycle. Since knowledge can be acquired iteratively, it makes sense that it should be validated iteratively. It is expected that as the knowledge base is developed, it will be constantly validated to ensure that the knowledge acquisition process is working properly. This type of validation also lends a sense of confidence to the user, as they can see correct solutions being produced by the knowledge base. Validating the final knowledge base will ensure that the knowledge is as complete as needed and as correct as possible given the requirements. As the knowledge base enters the maintenance portion of its lifecycle, additional benefits of BVAL's methodology are seen. Reports of incorrect solutions can be directly incorporated into the requirements of the knowledge base. The knowledge base can then be re-validated with the new requirements; BVAL will automatically correct the knowledge base, or guide the user toward areas of unacceptable incompleteness.

Which Methodology: As mentioned earlier, the only feasible methodology currently available that can formally compare a knowledge base against its requirements is validation by testing. BVAL uses this methodology throughout its lifecycle.

What Criteria: Currently, the measurement criteria used by BVAL is similar to that used by neural networks; a simple percentage, calculated by dividing the number of correct test cases by the total number of test cases. This criteria will give an accurate measurement of the knowledge base at any time during its development. The exact accuracy percentage desired is defined by the user; however, it is theoretically possible to make any BKB 100% accurate, with respect to a given set of requirements, by inputing enough knowledge to account for all possible combinations of evidence (i.e. a complete knowledge base). The intent of BKBs, however, is to allow for incomplete knowledge so that the same accuracy percentage can be achieved with less input from the user and less storage requirements for the knowledge base. Furthermore, recall that the entire KBS, should also be validated and verified. This system-wide KBS validation should include some metric relating to the expected long term reliability of the knowledge base; however, at this time that metric has not been formulated.

BVAL's Methodology

The approach taken by BVAL to validate a single test case is best described by a list of actions and tests followed by discussions of each alternative.

1. A test case containing evidence and the expected answer are submitted.
2. A query is run, returning a solution.
3. If the solution is correct (contains the expected answer) and no assumptions were made, that test case is considered validated, and the next test case is processed.
4. If the solution was correct, but contained assumptions, then the knowledge base is incomplete and needs to be corrected. (Note that the term "incomplete", used in this fashion with reference to a BKB, means "unacceptably incomplete in a specific area highlighted by the test case". This is not meant to imply that the knowledge base should be made "complete".)
5. Otherwise, the solution was incorrect, making the knowledge base a candidate for reinforcement learning on this test case.

Once a candidate for reinforcement learning is identified, a second query is submitted with the test case evidence and the expected answer both provided to the inference engine as "evidence". This forces the inference engine to produce a solution containing the expected answer. If this solution, which is known to be correct, contains assumptions, then the knowledge base is incomplete and needs to be corrected. Otherwise, the knowledge base just needs to be reinforced to accommodate this test case. Once the reinforcement procedure is complete, the validation proceeds to the next test case.

If the knowledge base was found to be incomplete, the user is given the option of bypassing this test case,

or attempting to correct the knowledge base. If the user wishes to correct the knowledge base, they are provided different means of viewing the problematic knowledge in order to efficiently deduce the remedy. Once the user has decided upon a remedy, they are allowed to edit the knowledge base using the familiar MACK tools. Upon completion of editing, the original test case is run again. This second check ensures the corrections made actually work–providing immediate feedback to the user. Since this second pass is handled like any other test case, if the knowledge base is still incomplete, the user can continue to edit. Additionally, this format allows the user to make iterative corrections to the knowledge base without altering the test suite.

Once every test case in the suite has been processed, all cases are passed through again, this time for metric purposes (each case is either correct or incorrect with no additional processing). The percent correct is considered the current accuracy level of the knowledge base. Since the reinforcement learning caused by one test case can impact the validation status of another test case, the entire test suite is run again (as the first time with reinforcement and interaction as necessary) with the test cases in a random order. The constant changing of the ordering of test cases is a technique borrowed from neural networks and is important here since order can affect validation.

After going through this process a number of times, the knowledge base should start to show a fairly constant level of accuracy. If this level of accuracy is acceptable to the user, then the knowledge base is considered validated for this set of requirements. If the accuracy is not acceptable, then further analysis is warranted. After all of the test cases causing incomplete errors have been corrected, BVAL should be running without user interaction; i.e., reinforcement is the only corrective action being taken. At this point, if the knowledge base is not achieving an accuracy of 100%, then thrashing is occurring between test cases indicating areas of incompleteness that need to be corrected. After the thrashing errors have been corrected, one final pass through the test suite should show an accuracy of 100%—indicating the knowledge base is validated against its requirements.

Test Cases

Application BVAL uses the MACK tools the user is familiar with to perform test case solicitation. Here the user is being prompted for the evidence portion of the test case used for Figure 4.

```
Please enter the EVIDENCE
Please pick a component:
  1 - Acute Bronchitis
  2 - Chronic Bronchitis
  3 - Cough
  4 - CXR
  5 - History and Physical
  6 - Interstitial Fibrosis
  7 - Malignant Disease
  8 - Obstructive Pulmonary Disease
  9 - PFTs
  10 - Pneumonia
  11 - Sputum or Tissue Culture
  0 - Abort
Choice: 3
Please pick an instantiation of Cough:
  1 - Cough = TRUE
  0 - Abort
Choice: 1
```

Although this example shows only one RV-state (instantiation) being submitted as evidence; the answer and evidence can contain many elements. Following the entering of evidence, some calculations are performed to find the connected/relevant region of the knowledge base (See (Gleason 1996) for more details.). The expected answer must be contained in this region; this quality being guaranteed during solicitation. Additionally, a check for potentially thrash-causing test cases is performed as each expected answer is submitted.

Following successful completion of test case solicitation, the validation process continues as outlined above. Details regarding the test cases are presented in the remaining sub-sections.

Numbers Ideally, the test suite (requirements) would contain every possible query that could be submitted to the knowledge base; however, in all but the smallest of domains this is impossible. It is possible to generate all combinations of evidence automatically; however, the user must provide an expected answer for each combination to complete the requirements. Clearly, this approach is infeasible. Not only would a combinatoric number of test cases be generated, but a large subset of these cases would be meaningless and unrealistic for the domain. Barring such an exhaustive listing, the test suite should contain those cases representative of the entire domain.

Ensuring the requirements are representative of the domain is the same problem the neural network community must solve in relation to the testing and training sets used on neural nets. Unfortunately, this is an extremely difficult problem. Even if a neural network is trained properly and passes all of its validation tests, there is still no guarantee that the network will ever solve a new problem correctly. Extensive testing, if the data is available, provides some indication; however, all testing is based on the assumption that

the available data correctly represents the problem domain.

Applying the lessons learned from the neural network community: data is good, more data is better. For neural networks this is not always true; data points corresponding to exceptions can detract from the accuracy of the net. For BKBs, however, if an exception should be accommodated by the knowledge base, it should be included in the test suite. This inclusion may result in an interactive session during validation, but BVAL will ensure that the exception is accounted for without detracting from the "non-exception" test cases. Bottom line, as many realistic test cases as possible should be included in the requirements.

Reinforcement Learning

Application In this portion of its methodology, BVAL compares the incorrect solution generated by the evidence to the known correct solution generated by the evidence and the expected answer combined. Recall that these solutions are actually paths through the knowledge base, and only one state can be active for any RV in a solution. The two solutions' joint probabilities will be different, with the correct solution's probability less than or equal to the incorrect solution's. This known ordering of the solutions' probabilities is used to compute the "reduction factor". This factor is the percentage the incorrect solution's probability would have to be reduced to be lower than the correct solution's probability. Every probability (S-node) in the knowledge base, except the ones associated with the correct solution, are then reduced by this factor. After this reduction is applied, an up-scaling factor is applied to all probabilities in order to prevent precision errors. Once this up-scale factor is applied, the reinforcement process is complete. If the same test case was re-run immediately, the correct solution would result.

Figure 4 shows the progression of changing probabilities during a reinforcement learning process. This example is derived from a decision tree on respiratory disorders presented in (Healey & Jacobson 1990). Initially, the query is submitted with evidence of Cough = TRUE. (Table 1 shows the correspondence between the alphanumeric RV-states and their more descriptive terms.) This query returns the solution amounting to the left-hand branch of the graph.[3] Since this solution does not contain the expected answer of $B = 1$, another query is submitted with $\{B = 1, C = 1\}$ as evidence. The solution from this second query is guaranteed to be correct, and since it does not contain any assumptions, the reinforcement process is started. After the reduction factor is applied, the up-scaling factor is derived, coincidentally from $B = 1$, then applied. The knowledge base ends up in the state denoted by the final probabilities. It is apparent that a re-submission of the initial query with $C = 1$ as the evidence would result in the correct answer.

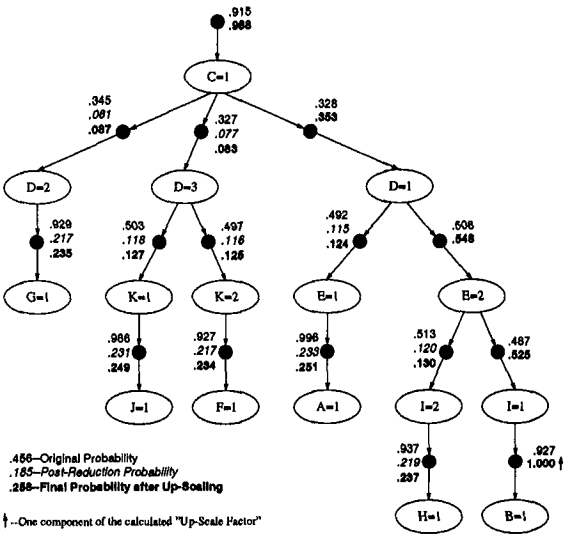

FIGURE 4. **Reinforcement Learning.** The probabilities change as reinforcement learning is applied to this knowledge base. Here the evidence was $C = 1$ and the expected answer was $B = 1$. Table 1 shows the translation of the alphanumeric RV-states to their more descriptive terms.

Why percentages? Using percentages to alter probabilities keeps all the magnitudes in the same relative position. For example, $P = 0.6$ and $Q = 0.3$ are reduced by 50% resulting in $P = 0.3$ and $Q = 0.15$. Even after the reduction, P is still double Q; the relative magnitudes are identical. Reducing the original probabilities by a constant, say 0.4, results in $P = 0.2$ and $Q = -0.1$. Not only are P and Q now of different relative magnitudes, but Q is negative; a practical impossibility and a consistency violation. If some error checking is done to prevent $Q \leq 0$, a side-effect results in some initially different probabilities being equal after error checking and correction. This side-effect actually changes the knowledge, an unacceptable result.

BVAL needs to guarantee that the correct solution will be the most probable after application of the reduction factor. To guarantee this quality, every individual probability is reduced (in percentage) by the amount that the overall joint probability needed to be reduced. This technique guarantees the original solu-

[3] The full solution is {Cough = TRUE, CXR = Abnormal with Nodular Infiltrate, Malignant Disease = TRUE} or $\{C = 1, D = 2, G = 1\}$.

RV Name		RV State	
Alpha	Descriptive	Numeric	Descriptive
A	Acute Bronchitis	1	TRUE
B	Chronic Bronchitis	1	TRUE
C	Cough	1	TRUE
D	CXR (Chest X-Ray)	1	Normal
		2	Abnormal with Nodular Infiltrate
		3	Abnormal with Diffuse Infiltrate
E	History and Physical	1	Infection
		2	No Infection
F	Interstitial Fibrosis	1	TRUE
G	Malignant Disease	1	TRUE
H	Obstructive Pulmonary Disease	1	TRUE
I	PFTs (Pulmonary Function Tests)	1	Normal
		2	Abnormal
J	Pneumonia	1	TRUE
K	Sputum or Tissue Culture	1	Positive
		2	Negative

TABLE 1. **Mapping alphanumeric names to descriptions** This table shows the correspondence between descriptive names and and alphanumeric names in the example associated with Figure 4.

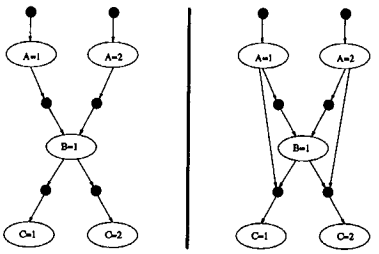

FIGURE 5. **Thrashing.**Here the BKB in part (a) will thrash when conflicting test cases are presented. Part (b) shows one possible remedy to the thrashing.

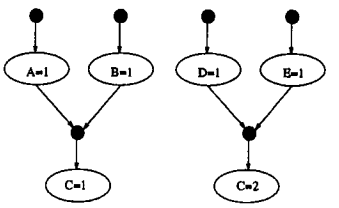

FIGURE 6. **Potential Thrashing.**This BKB is susceptible to thrashing if a test case like: Evidence: $A = 1$, $B = 1$; Expected Answer: $C = 2$ is included in its requirements

tion and every intermediate solution are less than the correct solution.

Thrashing As mentioned previously, due to the incompleteness allowed in BKBs, thrashing can occur during the validation. When thrashing occurs, the knowledge base oscillates (thrashes) between two or more states, where each state is lacking some of the qualities (accuracies) of at least one other state. For an example, see Figure 5. In this example thrashing occurs in the BKB represented by part (a) when the following two test cases are included in the requirements:

1. Evidence: $A = 1$—Expected Answer: $C = 1$
2. Evidence: $A = 2$—Expected Answer: $C = 2$

Validation of the first test case ensures that the following inequality is true: $P(C = 1 \mid B = 1) > P(C = 2 \mid B = 1)$. This must be the case since only one instance of C can be included in any solution. Conversely, validation of the second test case ensures: $P(C = 1 \mid B = 1) < P(C = 2 \mid B = 1)$. Left unchecked, this thrashing would continue indefinitely. A skillful user may detect this condition and choose to remedy the situation, or pick the best of the thrashing states. However, automatic detection of this condition is possible by monitoring the changes occurring in the knowledge base each time the test suite is validated. Once the test cases causing thrashing are identified, an analysis of the solutions generated by the test cases can provide pointers to the areas responsible for the thrashing. Figure 5(b) shows one possible remedy to this particular situation. Future work in this area will attempt to isolate and define conditions leading to thrashing, with possible automated or semi-automated correction.

While the aforementioned example showed a thrashing situation detectable during validation, it is possible to detect some potentially thrash-causing test cases before any validation is done. The BKB shown in Figure 6 is susceptible to thrashing if the following test case is present in its requirements:

1. Evidence: $A = 1$, $B = 1$—Expected Answer: $C = 2$

Validation of this test case alone would not result in thrashing; however, if a test case such as

2. Evidence: $D = 1$, $E = 1$—Expected Answer: $C = 1$

were added to the requirements at a later time, thrashing would result. BVAL checks for this potentially thrash-causing relationship between the evidence and the expected answer, warning the user appropriately.

Conclusions

This research develops a methodology and an automated tool for the validation of probabilistic knowledge bases. This tool, BVAL, incorporates aspects of both rule-based and neural network validation techniques to provide automatic correction and localization of problematic knowledge. The validation is accomplished by a formal comparison of the knowledge base against its requirements; where the requirements are composed of test cases.

The requirements (test suite) are processed one test case at a time, ensuring the knowledge base is as complete and as correct as needed to validate each requirement. When a requirement highlights areas of incompleteness, BVAL intelligently interacts with the user, assisting in the correction of the problem. If BVAL locates a requirement that can be satisfied by an adjustment of the probabilities, the adjustment is made automatically. This adjustment accounts for the strict probabilistic nature of the knowledge base and the finite resolution of any machine without bothering the user. At completion of validation, BVAL guarantees the knowledge base has remained consistent and is now 100% accurate with respect to the requirements.

BVAL is designed to integrate with the previous tools developed for Bayesian Knowledge Bases (Santos & Banks 1996) in PESKI, and to operate in an iterative manner. This incremental validation allows the user to gain confidence in the knowledge base as more and more knowledge is incorporated. After the knowledge base enters the maintenance portion of its lifecycle, BVAL remains useful by validating changes and additions to the knowledge base and/or its requirements.

References

Botten, N.; Kusiak, A.; and Raz, T. 1989. Knowledge bases: Integration, verification, and partitioning. *European Journal of Operational Research* 42:111–128.

Buchanan, B. G., and Shortliffe, E. H., eds. 1984. *Rule-Based Expert Systems.* Addison-Wesley Publishing Company.

Davis, R. 1979. Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence* 12:121–158.

Giarratano, J., and Riley, G. 1994. *Expert Systems: Principles and Programming.* PWS Publishing Company, Boston, MA, second edition.

Gleason, H. T. 1996. Probabilistic knowledge base validation. Master's thesis, Graduate School of Engineering, Air Force Institute of Technology.

Gonzalez, A. J., and Dankel, D. D. 1993. *The Engineering of Knowledge-Based Systems.* Prentice Hall.

Guida, G., and Mauri, G. 1993. Evaluating performance and quality of knowledge-based systems: Foundation and methodology. *IEEE Transactions on Knowledge and Data Engineering* 5(2):204–224.

Gupta, U. G., ed. 1991. *Validating and Verifying Knowledge-Based Systems.* IEEE Computer Society Press.

Healey, P. M., and Jacobson, E. J. 1990. *Common Medical Diagnoses: An Algorithmic Approach.* W. B. Saunders Company.

Hecht-Nielsen, R. 1991. *Neurocomputing.* Addison-Wesley Publishing Company.

Nazareth, D. L. 1993. Investigating the applicability of petri nets for rule-based system verification. *IEEE Transactions on Knowledge and Data Engineering* 4(3):402–415.

O'Keefe, R. M.; Balci, O.; and Smith, E. P. 1987. Validating expert system performance. *IEEE Expert* 81–89.

Pearl, J. 1991. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, San Mateo, CA. (Revised Second Printing).

Santos, Jr., E., and Banks, D. O. 1996. Acquiring consistent knowledge. Technical Report AFIT/EN/TR96-01, Department of Electrical and Computer Engineering, Air Forc e Institute of Technology.

Santos, Jr., E., and Santos, E. S. 1996. Bayesian knowledge-bases. Technical Report AFIT/EN/TR96-05, Department of Electrical and Computer Engineering, Air Force Institute of Technology.

Yager, R. R., and Larsen, H. L. 1991. On discovering potential inconsistencies in validating uncertain knowledge bases by reflecting on the input. *IEEE Transactions on Systems, Man, and Cybernetics* 21(4):790–801.

Zlatareva, N. 1994. An effective logical framework for knowledge-based systems verification. *International Journal of Expert Systems* 7(3):239–260.

MACK: A Tool for Acquiring Consistent Knowledge Under Uncertainty

Eugene Santos Jr.[1] **and Darwyn O. Banks and Sheila B. Banks**
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-7765
esantos@afit.af.mil

Abstract

We develop a new methodology and tool for knowledge acquisition under uncertainty. A new knowledge representation called Bayesian Knowledge Bases provides a powerful key to our approach and is well-grounded in probability theory. In this paper, we demonstrate the ease and flexibility with which knowledge acquisition can be accomplished while ensuring the *consistency* of the knowledge base as data is both acquired and subsequently maintained. We present the MACK tool and apply it to NASA's Post-Test Diagnostics System which locates anomalies aboard the Space Shuttles' Main Engines.

Introduction

Knowledge engineering new domains remains a daunting task for both the expert and the engineer involved. The knowledge must be elicited from the expert and then converted into a form according to the internal knowledge representation of the target expert system. Furthermore, the knowledge itself must then be validated and verified to ensure the system's reliability. Figure 1 lays out the standard three phase knowledge acquisition process.

The ease with which we perform a given phase is intricately tied to each of the other phases. For example, the interview should ideally be as painless as possible avoiding problems such as redundant questioning, overly rigid response templates — requiring the expert to answer using an inflexible and often unrealistic format, etc. Only the most intuitive and flexible of knowledge organization should be required of the expert. Unfortunately, if the internal knowledge representation is significantly different from the organization of the interview, then the knowledge engineer is faced with the onus of properly encoding the information. This typically entails a radical re-structuring of the given information while simultaneously attempting to preserve its original content.

[1]This research was supported in part by AFOSR Project #940006.

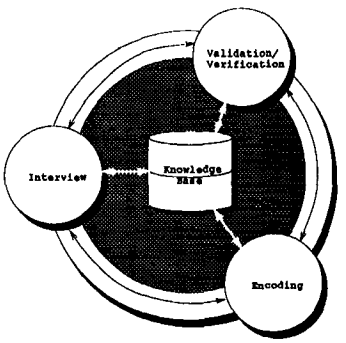

FIGURE 1. Knowledge acquisition process.

For the moment, let's assume that our knowledge representation is also relatively simple and mirrors the interview organization. While this simplifies the job for the knowledge engineer, we end up impacting our last phase. Clearly, there is a tradeoff between the amount of organization and flexibility inherent in our knowledge representation versus our ability to perform verification and validation over it. For example, the problem of consistency is especially sensistive. Without much organization, it is nearly impossible to detect when and where an incosistency has occurred.

There are many other factors that determine the success or failure of a knowledge acquisition tool. This includes many pragmatic concerns such as learnability of the tool, input requirements — user interface, and knowledge induction (Gaines & Shaw 1993; Gonzalez & Dankel 1993). All of the above factors serve towards building an ideal tool for assisting knowledge engineers.

The major difficulty faced by all knowledge engineers is in dealing with uncertainty: uncertainty in the expert's themselves about their knowledge and uncertainty in the engineer trying to translate the knowledge. Although it seems that all knowledge can be