Expert Systems With Applications, Vol. 6, pp. 57–66, 1993
Printed in the USA.

0957-4174/93 $5.00 + .00

Verification and Validation of Case-Based Systems

DANIEL E. O'LEARY

School of Business, University of Southern California, Los Angeles, CA

Abstract—*Verification and validation of artificially intelligent systems has been the focus of substantial recent research. However, little attention has been given in the literature to verification and validation for case-based systems. The unique structure of case-based systems is used to raise new* validation *issues, develop new approaches to generating comparative solutions for validation purposes, and investigate new approaches for examining the quality of the case base. In addition, this article presents new statistical and structural approaches designed to exploit unique aspects of case-based reasoning for* verification *purposes.*

1. INTRODUCTION

VIRTUALLY ALL THE RESEARCH in verification and validation has been focused on rule-based systems rather than other knowledge representations, such as case-based systems (Gupta, 1991). Rule-based systems make their judgments from rules, while case-based systems make their judgments from previous cases or experience.

Thus, the purpose of this article is to summarize existing approaches, develop new approaches, and raise new concerns for the verification and validation of case-based systems. To accomplish those activities, some of the unique factors of case-based systems are elicited. Those factors are then used as a basis for developing new approaches for verification and validation.

1.1. Role of Verification and Validation

Verification tests are aimed at "building the system right," and validation tests are aimed at "building the right system." Thus, verification examines issues such as ensuring that the knowledge in the system is represented correctly, while validation examines procedures to ensure the system makes correct decisions.

Verifying and validating play critical roles in the development and implementation of case-based systems. *A priori,* if the system is not verified then there may be errors in the case representations. If the system is not validated, then it may not make the desired quality of decisions.

1.2. Basis of Verification and Validation

Verifying and validating are accomplished by comparing what is expected to what is present. In verification, the basis for these comparisons is the knowledge representation and the knowledge stored in those representations. Typically, the case knowledge is represented using, for example, frames. Thus, these comparisons may include, for example, the structure of the cases (e.g., number of slots and number of slots filled with meaningful information), the structure of the interaction of the cases (e.g., frames typically use tree structures), and statistically based expectations (e.g., distribution of various case parameters). These are each discussed in more detail later in the article.

Learning mechanisms can impact the case-based system's performance and, thus, validation results. As a result of those learning processes, typically, case-based systems add cases to the case base as the system learns. Thus, an important issue in validating case-based systems is the impact on future system performance of adding cases from current experience to the case base for future use. Critical questions include, does comparative performance change based upon the order in which cases are added to the case base?

In validation, often human experts are the comparative basis. However, experts can be expensive or unavailable. Thus, one of the focuses of this article is the generation of alternative sources for comparison. Mathematical programming and statistics are developed as sources of comparative solutions.

Examination of the cases in the case base is a critical

Portions of this article were developed while visiting Bond University, the School of Computing Science, Gold Coast, Queensland 4229, Australia.

Requests for reprints should be sent to Daniel E. O'Leary, School of Business, University of Southern California, Los Angeles, CA 90089-1421, USA.

concern to the quality of the decisions and, thus, to validation. This article uses statistical and similarity measures to examine the quality and diversity of the case base.

1.3. This Article

This article proceeds as follows. Section 2 provides more detailed definitions and discussions of verification and validation. Section 3 reviews the previous literature on verification and validation of case-based systems. Section 4 argues that case-based systems are different than other types of systems and thus should require a unique approach to verification and validation. These factors provide the basis of the remainder of the article. Sections 5 and 6 discuss *verification* issues. Section 5 analyzes some domain-independent statistical approaches for the verification of case-based systems. Section 6 investigates some structural-based approaches that can be used to verify case-based systems. Sections 7–11 present *validation* approaches developed for case-based systems. Section 7 finds that the sequential addition of cases can cause the development of different solutions to the same problems, with different sequential processing of the cases. This is critical to both development and verification and validation of case-based systems. Alternative methods of generating solutions can be useful when expert time is limited, costly, or ineffective. As a result, Section 8 presents a mathematical programming approach that can be used to generate solutions to compare to those of case-based systems. Section 9 discusses the use of statistical models, also to generate comparable solutions. Test cases can be used to assist in the validation of the system. Thus, Section 10 presents an approach using genetic algorithms to generate test cases. The diversity of the cases in a case base can impact the quality of solutions generated. Thus, Section 11 develops approaches to test the similarity of cases so that the diversity of the cases in the case base can be assessed. Finally, Section 12 provides a brief summary of the article.

2. VERIFICATION AND VALIDATION

This section presents some definitions of verification and validation. Those definitions are then briefly analyzed in terms of what they mean for case-based systems.

2.1. Verification

Verification was defined by Adrion, Branstad, and Cherniavsky (1982) as "the demonstration of the consistency, completeness and correctness of the software." This definition often is supplemented to include redundancy to provide greater specificity to the notion of completeness.

Verification's dependence upon software indicates that the specific nature of case-based systems needs to be elicited to perform verification. Because verification is software based, that is, for case-based systems, verification is concerned with exploiting the software representations of, for example, cases and relationships between cases to establish tests for consistency, completeness, correctness, and redundancy.

Consistency, in case-based systems refers to parallel implementation of parallel structures, whether those structures are words or relations between cases, such as trees. Completeness is concerned with the possibility that knowledge or cases are omitted. Correctness refers to determination of whether or not there are any ascertainable errors in the knowledge, for example, circularity in a structure that is supposedly acyclic. Redundancy addresses the issue of duplication of knowledge, for example, duplicate versions of the same case.

Verification also can be dependent upon the domain. By exploiting knowledge of the domain, we can verify the knowledge in a case-based system. Consider the example of a system that has a number of cases that are different living rooms. Metaknowledge could be used to examine the cases to determine that there was something incorrect with a case that had a bathtub in the living room. There probably would be redundant knowledge in the case if the living room case contained two couches. Similarly, the case of a living room without a couch probably could be incomplete. Finally, it would be inconsistent if in the case two physically identical items were labeled with the same name, for example, couch. Such domain-dependent approaches are outside the scope of this article because they require specification of metaknowledge from the specific domain.

2.2. Validation

Adrion et al. (1982) indicate that validation is the determination of the correctness of the final program or software produced from a development project with respect to the user needs and requirements. In many software development projects, the needs and requirements can be established *a priori*. However, case-based systems are used in situations where the problem is not well structured enough to develop a rule-based system. Thus, for case-based systems there are likely to be few situations where specifications can be elicited *a priori*.

As a result, validation may have to employ different bases of comparison rather than requirements. The process of validation for case-based systems may be similar to the generation of the needs and requirements for other artificial intelligence (AI) systems. For example, validation may take the form of comparing an expert to the system for different test cases. Alternatively, there may be other approaches that could exploit the unique characteristics of case-based systems to

generate comparison bases. These approaches could be cost beneficial by limiting the use of experts in validation processes. The development of alternative models of comparison is a primary focus of this article.

3. VERIFICATION AND VALIDATION OF CASE-BASED SYSTEM: PREVIOUS STUDIES

One purpose of this article is descriptive. This section summarizes some of the previous descriptions of verification and validation efforts of case-based systems. Unfortunately, there have been a limited number of studies summarized in the literature (the literature on case-based systems is also limited, in general). In addition, in many situations the discussion of those efforts is limited to a few sentences or paragraphs. Further, typically the focus of those discussions has been on validation. There are virtually no descriptions of verification efforts for case-based systems.

However, it is not surprising that few have discussed verification and validation efforts. The purpose of much of the research on case-based systems, to date, has been to explore various developmental aspects of case-based systems, such as retrieval or indexing. Thus, the lack of research on verification and validation is not a criticism. Instead, this indicates the importance of developing approaches to assist in both the verification and validation of case-based systems.

3.1. Protos

Possibly the most extensive validation of a case-based system in the literature is for the system Protos. Protos (Bareiss, 1989; Bareiss, Porter, & Weir, 1988) is a case-based learning apprentice that learns to perform heuristic classification under the guidance of a human expert. The evaluation of Protos was in the area of clinical audiology, where it learned to classify hearing disorders from featural descriptions in terms of patient symptoms, history, and test results.

Because Protos is a classification system, the validation considered the accuracy of its classifications. Accuracy comparisons included comparison to other machine learning approaches and to human experts and students.

When compared to other forms of machine learning, Protos was apparently very successful, given the set of test problems. The accuracy of Protos was found to be far superior to the well-known ID3 (Quinlan, 1986) and Cobweb (Fisher, 1987). On a set of 26 test cases, Protos was correct 100% of the time, ID# 29% of the time, and Cobweb 58% of the time.

When compared to human experts and students, the system also did well. While Protos was 100% correct on the test problems, two clinicians had 92 and 81% correct, while students had a mean of 73%.

Apparently, Protos faced a reasonably large base of cases. In the training of Protos, a human expert characterized 7 of the final 50 training cases as unusual.

3.2. HYPO

One of the best-known case-based systems is HYPO. HYPO was a system developed by Ashley and Rissland (1988) to analyze trade secret law. The reported validation of HYPO's (Ashley & Rissland, 1988) performance was the comparative analysis of a single "real-world" case by the system. They compared HYPO with what the court did on that case and found that the system agreed with the court.

3.3. Clavier (Mark, 1989)

The description of evaluation efforts for Clavier is actually a plan for validation. However, that plan had at least two components. First, the adaptation of the cases by the system would be compared to experts. Thus, one of the components of the system would be tested, probably using a classic Turing test. Second, once the system was "working" the overall behavior of the system would be compared to experts for quality of recommendations.

3.4. The Battle Planner (Goodman, 1989)

To validate The Battle Planner, a randomly selected set of 10% of the cases were put aside before indexing. Then, those cases were treated as hypothetical battle situations. Predictions were made regarding the outcome of those situations. Using this approach, the system was found to be 81.3% accurate in the prediction of the victor of the case for land warfare.

The validation efforts pointed toward an interesting finding. Goodman found that the more cases that were retrieved for analysis of a hypothetical case the better the performance of the system. Such a finding may occur for a number of reasons. First, it may be that the more cases offering feasible solutions the more likely that the solution space is spanned. Second, it may be that the more feasible solutions the more likely that a directly similar case could be found. In either case, this provides additional evidence about the importance of the quality of the cases in the case base on the performance of the system.

3.5. Summary

An analysis of the published reports of the validation of case-based systems indicates that the approaches used to date are similar to those used for other intelligent or expert systems. Typically, the validation of each system has used the comparison of the system to human experts or machine learning. This comparison ranged from a single case to multiple test cases. In gen-

eral, the system has been compared to, for example, human experts.

4. SOME UNIQUE FACTORS OF CASE-BASED SYSTEMS

The focus of this section is to elicit some of the unique aspects of case-based systems and use them as a basis for eliciting different approaches and concerns to verification and validation. The existence of unique factors both indicates a need to establish new approaches and provides new opportunities.

As noted by Ashley and Rissland (1988, p. 70), case-based reasoning is used ". . . to capture expertise in domains where rules are ill-defined, incomplete or inconsistent." It can be difficult to investigate the performance of a system in the presence of such difficulties. This suggests that it may be difficult to establish expectations or standards of behavior in case-based systems. In particular, the well-established process, in traditional software engineering, of using *a priori* specifications may not be feasible.

Because cases typically are represented using frames rather than rules, this indicates that verification approaches for case-based structures could exploit the frame structure. In particular, for example, verification processes could exploit the nature of cases within the context of the frames. This is important because virtually all of the work on verification is for rule-based systems.

In addition, case-based systems are unique compared to other types of AI systems in that they often add solved cases to their case base. Previous solutions become part of their experience. Those cases are then used in the development of future solutions. This is a critical difference from, for example, rule-based systems, where the knowledge base in general is static.

Although case-based systems often are designed to create new solutions, typically the quality of the solutions is dependent, in large measure, upon the case base. The quality of the recommendations of a case-based system is dependent upon the quality and quantity of the cases in the case base.

In general, a case-based system will be able to generate a better recommendation if it has a larger rather than smaller case base (e.g., Ruby & Kibler, 1988; Goodman, 1989; Gaines, 1991). It is likely that with a larger base of experience the situations faced by the system will have been seen before and thus a case will be available to assist in solution of the problem. Even if no identical situation has been seen before, the system is likely to have solutions that are so-called "near miss" situations.

If the cases are highly correlated, then the system will be limited in the diversity of solutions it can generate. Thus, it is not unusual that Bradtke and Lehnert (1988, p. 91) find ". . . that the most dramatic factor influencing the effectiveness of a case base is the number of unique problem states underlying the case base encoding."

Some of these issues can be addressed using approaches such as the direct analysis of the cases by experts, comparison of expert solutions to the system, and investigation of the system reasoning (choice of cases) by experts. However, the focus of this article is on the development of alternative approaches to account for these unique factors.

5. DOMAIN-INDEPENDENT VERIFICATION OF CASES: STRUCTURAL APPROACHES

In this section, no assumptions are made of the domain. Instead, the individual structure of the cases and the structural relationship between the cases are used to provide some bases with which to verify the cases. Unlike the next section, which uses statistics to identify anomalous cases that appeared to be in error, this section uses that structure to identify cases that are in error. Particular emphasis is placed upon using those structures to design verifiable systems or systems that mitigate the opportunity for the existence of the types of errors isolated.

5.1. Consistency

Errors in consistency can occur for many reasons. For example, people can make errors by misspelling or using different names for the same object, actor, or activity.

To design a system to mitigate those types of errors, at the initiation of a new case-based system the user could be required to list each of the cases and corresponding case attributes before any data is entered. Then, those lists of feasible entries for each of the cases and attributes would be used to fill attribute slots. Developers of different cases could choose from the lists of attributes for the content for a specific attribute. As seen below, these lists also can be used for other purposes.

5.2. Redundancy

Redundancy errors occur if the user is able to develop the same case more than once. The situation of redundant cases can cause difficulties, primarily in maintenance situations. One version of the case could be revised while another is not. This could cause confusion and possibly errors if the unrevised case was used by the system. The possibilities of redundant cases can be mitigated if the user is required to establish a list of the specific cases prior to actually establishing the data within each case. As new cases are added, they

would be added to the list of feasible cases. Each case would then be referenced by its specific name. Thus, a single version of any given case would be permitted.

Redundancy errors also occur if the user is able to enter the same attribute information into two or more attribute spaces for a given case. There are two basic situations: two attribute instances in the same attribute slot and the same two attribute instances in different slots in the same case. In the first situation, maintenance of one entry but not the other may cause ambiguity as to the proper contents. In the second situation, the attribute instance may be an inappropriate occurrence for that attribute instance.

The first situation could be eliminated if each attribute slot has space for only one entry. The second situation could be eliminated if the attribute instances could only come from eligible lists, as discussed earlier.

5.3. Completeness

There are at least two types of completeness errors in verification: missing cases and missing attributes for cases. There are some steps that can be taken to mitigate these errors.

Completeness of the cases may be difficult to assess. However, having the users establish a list of cases they intend to enter into the system prior to entering data about the cases could facilitate knowing about completeness. Thus, if a case was on the list but not in the case base this could indicate an incompleteness error.

Completeness of the attributes may also be a difficult issue. There are at least two approaches to analysis of completeness. First, each case could be required to have slots for the same number of attributes. As an example of this, the two cases in Table 1 have different numbers of attributes. "TIME" is in the first but not the second.

Second, users could be required to indicate the number of attributes, of the total set, that would have information in them for a specific case. For example, in Table 1 for the first case seven attributes would be specified. If a specific attribute slot had no information for a given case, then a variable representing that alternative would be entered into the slot (e.g., "nil" as in Table 1).

Completeness tests could also make use of the lists developed for consistency. In addition, after the user put the attributes onto a list it would be possible to reconcile the lists and the completed cases to determine if each of the attributes put onto lists were used. If there was an attribute that was listed but not used, that could indicate that one of the cases omitted appropriate information.

Completeness in the verification of cases requires that the cases that are planned to be included in the system are included and that for each of the cases a complete case specification is given.

TABLE 1
Sample Cases

Name: M-MEAL774[a]
Isa: (⟨M-MEAL⟩)
Category: INDIVIDUAL
Slots
  ACTUAL-RESULTS: NIL
  CHARACTERS: (?HOST ?GUESTS ?PARTICIPANTS)
  CONSTRAINTS: (⟨C-LIMITED-SPACE778⟩)
  DEFINED SLOTS: NIL
  DESCRIPTOR: NIL
  EXPECTED-RESULT: NIL
  FOLLOW-UP: NIL
  GOALS: (⟨E-EAT776⟩ ⟨S-HUNGER777⟩)
  GUESTS: ⟨*JLK*S-GROUP⟩
  HOST: ⟨*JLK*⟩
  ORDER: NIL
  PARTICIPANTS: (?HOST ?GUESTS)
  SETTING: ⟨*JLK*S-HOUSE⟩
  STEPS: NIL
  TIME: NIL

Name: M-MEAL80[b]
Isa: (⟨M-MEAL⟩)
Category: INDIVIDUAL
Slots
  ACTUAL-RESULTS: ⟨ACTUAL-RESULT80⟩
  CHARACTERS: (?HOST ?GUESTS ?PARTICIPANTS)
  CONSTRAINTS: (⟨C-COST11⟩)
  DEFINED SLOTS: NIL
  DESCRIPTOR: ⟨MEAL-DESCRIPTOR80⟩
  EXPECTED-RESULT: ⟨EXPECTED-RESULT80⟩
  FOLLOW-UP: NIL
  GOALS: (⟨S-HUNGAR80⟩ ⟨E-EAT80⟩)
  GUESTS: ⟨*JLK*S-PARENTS⟩
  HOST: ⟨*JLK*⟩
  ORDER: (⟨SC-SALAD80⟩ ⟨SC-MAIN-COURSE80⟩)
  PARTICIPANTS: (?HOST ?GUESTS)
  SETTING: ⟨*JLK*S-HOUSE⟩
  STEPS: (⟨SC-SALAD80⟩ ⟨SC-MAIN-COURSE80⟩)

[a] From *Proceedings: Case-Based Reasoning Workshop* (p. 28) by J. Kolodner, 1988, San Mateo, CA: Morgan Kaufmann. Copyright 1988 Morgan Kaufmann. Reprinted with permission.
[b] From *Proceedings: Case-Based Reasoning Workshop* (pp. 29–30) by J. Kolodner, 1988, San Mateo, CA: Morgan Kaufmann. Copyright 1988 Morgan Kaufmann. Reprinted with permission.

5.4. Correctness

The role of graph theoretic structure permeates case-based reasoning systems. Ashley and Rissland (1988a) demonstrate two different "claim lattices" (in solution generation). The structure of those lattices is that of a tree. Navinchandra (1988) illustrates that the object hierarchy used for matching is a tree. The causal structures generated by the system discussed in Navinchandra (1988) are acyclic graphs. If it is known *a priori* that the solution or case relationships will be tree or an acyclic graph, then that structure can be exploited to ensure that the structure is correct.

There are a number of characteristics of trees that allow for rapidly determining if a structure is a tree. For example, each child has at most one parent. Another check is to ensure that a node in the solution

does not have an arc going from itself to itself, which would imply inheritance from itself. Alternatively, the structure may be that of an acyclic network. In that situation, the resulting structure must be free of cycles for the system to be correct.

6. DOMAIN-INDEPENDENT VERIFICATION OF CASES: STATISTICAL APPROACHES

Statistical approaches to domain-independent verification exploit statistical methods as the basis of establishing expectations. The general approach is to determine if there is any anomalous behavior exhibited in statistical summaries of characteristics of cases. Anomalies indicate that something may be wrong and, thus, should be further analyzed. The existence of anomalies is established by developing and using statistical distributions.

There are two different statistical approaches to finding anomalies. Their implementation in a specific system would require accounting for the specific set of cases. As a result, the analysis presented here is designed for the example cases presented here but can be generalized. To illustrate these approaches, consider the example case in Table 1.

6.1. Distribution of Slot Contents Per Case

Analysis of slot contents can provide insight into the existence of anomalous behavior. One approach is to develop a distribution of the number of slots in each case with, for example, nil contents. Then, the distribution could be analyzed to determine if any of the cases were exhibiting unusual behavior.

The first example case has 15 generic slots for cases, of which 8 have the value nil and 7 have values different than nil. At the extreme, if all cases but this case had, say, either 0, 1, or 2 values of nil then that would suggest that this specific case was unusual. Such an approach could help locate incomplete and underspecified cases.

6.2. Distribution of Contents Per Slot Across Cases

In addition, distributions of slot content could be developed, for example, a distribution of nil values for each slot could be developed. If there was a slot where the number of values nil was very large, then that could suggest anomalous behavior because it would indicate that slot is only rarely considered important. Because this approach would identify slots where nil was the value, it would suggest that those slots may be incomplete across many different cases. For example, both "FOLLOW-UP" and "DEFINED SLOTS" contain the value nil for both cases.

Another approach would be to determine distributions for slot contents for each individual slot. In the example, assume that there were 40 cases that had a nonnil value for host. If 39 were ⟨JLK⟩ and 1 was ⟨JKL⟩, then the case with the single occurrence would warrant further investigation. This could assist in the location of incorrectness problems.

6.3. Analysis

In each situation, these distributions could be analyzed as in the extreme cases listed here and using established forms of analysis. For example, outlier analysis could be used to find those points in those distributions that appear to come from different distributions and thus may deserve further investigation. In addition, statistical tests of significance (e.g., t test, etc.) could be used to determine anomalies.

7. SEQUENCE EFFECTS OF ADDING CASES TO THE CASE BASE

One of the critical conditions of validation is the ability to duplicate the response of a system under similar conditions. Unfortunately, the behavior of case-based systems can vary depending upon the order in which cases are processed.

Case-based systems use a number of approaches for choosing the best match from the case base for the solution of the case under consideration. After the system proposes a solution, it then adds the new case to its case base for further use. Validating these systems is difficult because, as shown in this section, the order in which validation test cases are added to the system can influence the system's proposed solutions.

7.1. Solutions Are a Function of Order of Cases Encountered

Consider a case-based system that adds cases to its case base as it solves them. Assume that one of the primary bases of solution is the number of attributes that the situation under consideration has in common with some other case in the case base. Suppose that two outcomes, x and y (captured in the first slot), are possible. Next, suppose that there are six additional slots that capture attributes of the individual cases. It is assumed that attribute 1 is in the first slot, and so forth.

Suppose that the initial case base consists of the two cases, 1. (x; a, b, c, d, f, m) and 2. (y; a, b, c, h, g, m). If the next case seen is 3. (-; a, b, c, d, e, n), then it is resolved as 3. (x; a, b, c, d, e, n). As result, if the next case is 4. (-; a, b, c, h, e, n) then it will be resolved as 4. (x; a, b, c, h, e, n).

However, if case 4 is seen before case 3 then the following scenario will occur. Case 4. (-; a, b, c, h, e, n) will be resolved as 4. (y; a, b, c, h, e, n), while case 3. (-; a, b, c, d, e, n) becomes 3. (y; a, b, c, d, e, n).

Clearly, the sequence can have a major impact on the system. Solutions developed by the system can be

an artifact of the order in which they are processed. This presents a major concern in validation. Although the use of test cases may find that the system agrees with experts or other models, this may be simply a fortuitous function of order.

7.2. Impact of Order Effects

Case-based systems add cases as they encounter them. Because the solutions they provide are, in part, a function of their case base, the addition of cases to the case base can change the solutions they provide. Thus, the order effect is not unexpected. However, the order effect discussed here is important to more than just the development and validation of case-based systems. Such order effects can impact the security of these systems. Someone interested in manipulating a case-based system could manipulate the order of a set of cases the system processes. After the cases have been added to the case base, a more desirable solution may be obtained by the manipulator. In the above example, assume y is "not grant loan" and x is "grant loan." Thus, someone interested in obtaining a loan would have an interest in the order in which cases were presented to the system.

8. MATHEMATICAL PROGRAMMING FOR COMPARATIVE SOLUTIONS

Mathematical programming, in particular, goal programming (e.g., Charnes & Cooper, 1977), can be used to solve some case-based decision problems. Solutions generated using this mathematical programming approach can be compared to solutions generated by the specific case-based systems to determine the quality of those systems as a means of validation. Such an approach can mitigate the need for the comparative use of expertise in the evaluation process. However, unlike a comparison to a human expert the quality of the mathematical programming approach largely is limited to the quality of the cases represented in the case base.

8.1. Mathematical Programming Approach

It is assumed that each case can be represented by a vector, $\mathbf{C}_i$ of m components, for $i = 1, \cdots, n$. Let $c_{i,j}$ represent the jth attribute of case i. For purposes of exposition, it is assumed that those attributes are numeric constants. This is not limiting because many logical relationships can be modeled using vectors of this type (e.g., Garfinkel & Nemhauser, 1972).

A decision variable is required to determine if a case should be retrieved to develop a solution. If case i is not chosen, then $x_i = 0$, while if case i is chosen then $x_i = 1$.

Let $\mathbf{A}$ be a vector of m components, where a_j is the jth component in that vector. The vector $\mathbf{A}$ can be used to represent the case situation the decision maker currently faces, for which a match in the case base is being sought.

Goal programming attempts to minimize the deviation from some case or combination of cases. It is done by minimizing the extent to which the solution found "deviates" from a desired solution. Let $e_j^+ \geq 0$ and $e_j^- \geq 0$ be "deviation" variables, where e_j^+ is the amount by which a solution component exceeds a_j and e_j^- is the amount by which a solution component is under a_j. Let those vectors be represented by the vectors $\mathbf{E}^+$ and $\mathbf{E}^-$.

The basic goal program can be represented as follows.

$$\text{Min} \sum_j e_j^+ + \sum_j e_j^-$$

$$\mathbf{c}_1 x_1 + \mathbf{c}_2 x_2 + \cdots + \mathbf{c}_n x_n = \mathbf{A} + \mathbf{E}^+ - \mathbf{E}^-$$

$$1 \geq x_i \geq 0 \quad (\text{for all } i), \quad e_j^+ \geq 0$$

$$\text{and} \quad e_j^- \geq 0 \quad (\text{for all } j).$$

This formulation establishes the deviation changes required to be made to the vector $\mathbf{A}$ to adopt it to the case base. This is the mirror image of the change that would be made to adopt a case to meet the needs of the situation, $\mathbf{A}$.

This formulation can be added to and generalized to meet both the different approaches to matching and the weighting of attributes that sometimes takes place in case-based systems.

8.2. Alternative Case Matching Approaches

The solution of the mathematical program provides a matching between combination of cases in the case base and the case at hand ($\mathbf{A}$). There are at least two different implementations of this formulation, resulting in different sets of constraints that would be added to the formulation depending upon the type of analysis used. First, the concern might be with choosing the one case that is most similar to the situation of concern. In that situation, the following constraints also would be added:

$$x_i = 0 \text{ or } 1 \quad (\text{for all } i) \quad \text{and} \quad \sum_i x_i = 1.$$

Second, the interest could be in the best combination of cases, where the combination is a convex combination. In that situation, the following constraints would be added:

$$1 \geq x_i \geq 0 \quad (\text{for all } i) \quad \text{and} \quad \sum_i x_i = 1.$$

8.3. More General Cases

The formulation in Sections 8.1 and 8.2 required cases with fixed parameters. However, that approach can be generalized to allow more "flexible" case representations. In particular, the vectors of constants $\mathbf{C}_i$ can be generalized to a vector of variables subject to a constraint. Mathematical programming use of this kind of structure is referred to as generalized linear programming (GLP) by Dantzig (1963).

For example, in GLP rather than numeric constants to represent a case variables and a constraint on those variables would be used. Thus, if case $\mathbf{C}_i$ had three attributes then it might be represented as $c_{i,1}$, $c_{i,2}$, $c_{i,3}$, where, for example, $c_{i,1} + 2*c_{i,2} + 3*c_{i,3} \leq \mathbf{R}_i$, where $\mathbf{R}_i$ is a constant. Values could be attributed to each of $c_{i,j}$ $(j = 1, 2, 3)$ subject to that constraint.

8.4. Extensions

In some situations, it may be that one deviation should be weighted more than another. In addition, it may be that deviations (+ or −) should be weighted differently. In these situations, a weight could be associated with each e_j^+ and e_j^- for each j.

The mathematical program discussed in this section can be generalized to deviations of the form $e_{i,j}^-$ and $e_{i,j}^+$. In addition, it can also be formulated to minimize the number of attribute deviations. These extensions could result in more complex formulations with more variables.

9. STATISTICAL ANALYSIS OF CASE OUTCOMES: CASE SOLUTION AND QUALITY

Statistical analysis can be used to provide a comparative model of the quality of the solutions presented and examine the reasonableness of the solutions attributed to cases in the case base. Statistical approaches can be used in the validation of cases by exploiting the information contained in the cases that relate to the attribute slots (independent variables) and the outcome of the specific case (dependent variable). Such an approach most likely would be useful in those situations where both dependent and independent variables have few occurrences or are continuous variables, for example, a case base for binary decisions, such as "guilty or not guilty" and "invest or not invest."

Statistical models are similar to case-based models: The addition of new data can change the relationship between the independent and dependent variables. As a result, to model the case-based system regression coefficients (and other statistics) may have to be recalculated whenever a new case is added to the case base.

9.1. Regression as a Comparable Model

A statistical model of the entire set of cases, such as regression (or a neural net model—their performances have been reported as similar), can be used to test the quality of the output of the system. For example, given the case base as a data set, regression analysis (in its many different forms, e.g., logistic regression) would be used to develop a set of coefficients. Then, for each new case situation encountered that set of coefficients could be used to estimate the outcome. That outcome would then be compared to the outcome of the system. If the outcomes were not the same, then that would be cause for further analysis to determine which is appropriate. For example, human experts may investigate the resulting differences.

9.2. Regression as a Test of Case Quality

If there are errors in the cases, then those errors will be used in the system, possibly generating additional erroneous system behavior. Regression also could be used to test the quality of the remaining cases. In this situation, a portion of the case base would be used to develop a statistical model to estimate the dependent outcome variable. Then, that model would be used on the remainder of these cases to predict the outcome of the cases based upon the independent variables in the specific cases. That prediction would be compared to the actual outcome to validate the specific case. Disagreement may signal the need for detailed examination of the case.

10. GENETIC LEARNING FOR GENERATING TEST CASES

Test cases are probably one of the most frequently used approaches in the validation of intelligent systems. They are a necessary part of the comparison of human experts to the system. Thus, their use in case-based systems is not unexpected. One approach to generate test cases is through the use of genetic learning algorithms.

Genetic algorithms employ a concept of adaptive efficiency in the context of a probabilistic search method (Goldberg, 1989). They are called genetic algorithms because they employ methods analogous to genetic transfer from one organism to its offspring. They assume that, as in nature, the organisms that are best suited to the environment flourish and produce offspring with similar genetic traits.

Genetic algorithms employ current-"generation" information (existing cases) to develop the next- and future-generation information (new cases). Typically, it is assumed that information is stored in a vector of components, where each element is either a 0 or a 1 (have the attribute/do not have the attribute). The

next-generation vector is reproduced by operations on the current set of vectors.

Genetic learning algorithms can be exploited to develop other generations of cases for test purposes. As above, it is assumed that a case i can be written as a vector $\mathbf{C}_i$, where there are m attributes in each case, $d_{i,j}$, where $j = 1, \cdots, m$. For illustrative purposes, it will be assumed $c_{i,j}$ equals either 0 or 1, although this can be generalized.

There are a number of operators that convert current-generation vectors into next-generation vectors. These include mutation and crossover. Mutation refers to the mutation of individual elements in a given case. For example, $c_{i,j}$ becomes "not" $c_{i,j}$. Crossover refers to merging elements from two different cases i and k. For example, $\mathbf{C}_i$ and $\mathbf{C}_k$ can be merged to create a new case with $(c_{i,1}, \cdots, c_{i,h}, c_{i,h+1}, \cdots, c_{i,m})$. The new case contains elements from both vectors.

This approach assumes that an outcome can be associated with the new cases (e.g., guilty or not guilty). In some situations (e.g., Deugo & Oppacher, 1989), algorithms can be used to establish a corresponding benefit or outcome. Alternatively, the associated outcome possibly could be constructed based upon the original parent vector outcomes or established by expert analysis.

11. QUALITY OF CASE BASE: SIMILARITY BETWEEN CASES

As noted earlier, one of the most critical aspects to system success can be diversity of cases in the case base. Thus, part of the validation of the case base can be aimed at an analysis of the relationship of similarity between different cases. This section presents four different approaches to measure the extent of similarity between cases.

11.1. Correlation Coefficient

Probably the best-known statistical measure of the relationship between two entities is the correlation coefficient. Assume that each of the attribute slots and solutions are numeric. In that situation, the correlation coefficient will provide a summary statistic for the similarity between two different cases. However, in many situations the attributes are not numeric. For those situations, we can develop other measures of similarity.

11.2. Counts

Next, assume that not every slot is numeric. Then, one measure of the similarity of one case to another is a numeric count of the number of attributes they have in common, normalized by the number of slots. Depending upon the situation, the simultaneous slot contents of nil may or may not be counted as an item of similarity. In the example cases, there are six attributes in common between the two cases, including nil responses. Assuming a 15-attribute case structure, the two cases are related at the 40% level.

11.3. Weighted Counts Between Cases

The last approach can be extended to weigh the slots differentially in terms of importance of the slot or in terms of difference of content. For example, "GUESTS" may be twice as important as "HOST." Further, it may be that the difference between nil and any other value is more than any two nonnil values. Possibly, expert understanding of individual case situations could be used to establish weights on the attributes and content differences. The development of such weights would vary between individual applications.

11.4. Cross-Attribute Counts

Similarly, the counts could be made across attribute. We could measure the extent of similarity as (number of occurrences of value "x")/(total number of cases). Thus, if each case had the same value x for an attribute then we would have that ratio equal to 1.

In the example, the "HOST" is common to each of the two cases. Thus, on that dimension the cases are the same: There is no diversity of HOST. Thus, a vector containing the percentage of each attribute would isolate those attributes that are similar across the case set.

11.5. Use of Similarity Measures

This section has suggested four similarity measures. Additional measures could be developed, for example, based upon semantic distances between slot contents. Ultimately, the use and meaning of those measures would vary from application to application based upon our expectations.

If the cases are "too" similar, then that can indicate any of a number of steps to be taken. First, it may indicate a need for a larger case base. This could require that some new cases that are not as similar as the existing cases be added to the case base. Second, in this situation it could indicate that the cases are underspecified. Additional attributes could be solicited for each of the cases that differentiate the cases from each other. Third, a high degree of similarity among the cases could indicate an error. In particular, a correlation of 1 would indicate identical contents, that is, redundancy of cases.

12. SUMMARY

This article has investigated verification and validation of case-based systems. Throughout, those methods were designed to exploit the unique factors of case-based

systems. The limited case-based verification and validation research was summarized and new methods were developed.

Two approaches to verification were discussed. Statistical approaches were elicited for the determination of cases that appeared to exhibit anomalous behavior. Approaches based upon the structure of a case and structural relationships between cases were developed to isolate those situations where there were errors in those representations.

The remainder of the article dealt with validation issues. An important criticism of case-based systems, in general, with implications for validation was captured in an analysis of the impact of sequence of cases processed and added to the knowledge base. It was found that sequence of processing cases can impact assignment of solution.

Then, two different approaches for generating comparative solutions were developed. First, a goal program was presented that used the representations of the existing case base to develop solutions. The models allowed for either a combination of cases or choice of the single most similar case. Second, regression analysis was presented as another model for generating comparable solutions to the case-based system.

One of the unique aspects of case-based systems are the cases. The quality of the system solutions is in general a function of the quantity and quality of the cases. Genetic algorithms were explored as a means of generating test cases. A statistical approach was proposed as a vehicle for examining the quality of the case base. Finally, measures for analysis of the similarity of cases within the case base also were investigated.

Acknowledgments—The author would like to thank the three referees for their comments on two earlier versions of this paper.

REFERENCES

Adrion, W., Branstad, M., & Cherniavsky, J. (1982). Validation, verification, and testing of computer software. *ACM Computing Surveys,* **14**(2), 159–192.

Ashley, K., & Rissland, E. (1986). Comparison and contrast: A test of expertise. [Reprinted in J. Kolodner (Ed.) (1988a). *Proceedings: Cased-Based Reasoning Workshop* (pp. 31–36). San Mateo, CA: Morgan Kaufmann].

Ashley, K., & Rissland, E. (1988b). A case-based approach to modeling legal expertise. *IEEE Expert,* **3**, 70–77.

Bareiss, R. (1989). The experimental evaluation of a case-based learning apprentice. In *DARPA* Eds. (pp. 162–167). Los Altos, CA: Morgan Kaufmann.

Bareiss, E., Porter, B., & Wier, C. (1988). Protos: An exemplar-based learning apprentice. In *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 12–23). University of California at Irvine, June 1987.

Bradtke, S., & Lehnert, W. (1988). Some experiments with case-based search. In J. Kolodner (Ed.), *Proceedings: Cased-Based Reasoning Workshop.* (pp. 80–83). San Mateo, CA: Morgan Kaufmann.

Charnes, A., & Cooper, W. (1977). Goal programming and multiple objective optimizations. *European Journal of Operational Research,* **1**(1), 347–362.

Dantzig, G. (1963). *Linear programming.* Princeton, NJ: Princeton University Press.

Davis, R. (1976). Use of meta knowledge in the construction and maintenance of large knowledge bases. Unpublished PhD dissertation, Stanford University, Stanford, CA.

Deugo, D., & Oppacher, F. (1989). Applications of case-based reasoning using knowledge base and genetic techniques. In *DARPA* (pp. 239–244). Los Altos, CA: Morgan Kaufmann.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning,* **2**, 139–172.

Gaines, B. (1991). The trade-off between knowledge and data in knowledge acquisition. In G. Patetsky-Shapiro & W. Frawley (Eds.), *Knowledge discovery in databases* (pp. 491–505). Cambridge, MA: AAAI/MIT Press.

Garfinkel, R., & Nemhauser, G. (1972). *Integer programming.* New York: Wiley Interscience.

Goel, A., & Chandrasekaran, B. (1988). Integrating model-based reasoning with case-based reasoning for design problem solving. In *Proceedings of the AAAI-88 Workshop on AI and Design.* St. Paul, MN.

Goldberg, D. (1989). *Genetic algorithms.* Reading, MA: Addison-Wesley.

Goodman, M. (1989). CBR in battle planning. In *DARPA* (pp. 264–269). Los Altos, CA: Morgan Kaufmann.

Gupta, U. (1991). *Verification and validation of knowledge-based systems.* New York: IEEE Computer Press.

Kolodner, J. (Ed.) (1988). *Proceedings: Case-Based Reasoning Workshop.* San Mateo, CA: Morgan Kaufmann.

Koton, P. (1989). Evaluating case-based problem solving. In *DARPA* (pp. 173–175). Los Altos, CA: Morgan Kaufmann.

Mark, W. (1989). Case-based reasoning for autoclave management. In *DARPA* (pp. 176–180). Los Altos, CA: Morgan Kaufmann.

Navinchandra, D. (1988). Case-based reasoning in Cyclops: A design problem solver. In J. Kolodner (Ed.), *Proceedings: Case-Based Reasoning Workshop.* (pp. 286–301). San Mateo, CA: Morgan Kaufmann.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning,* **1**(1), 81–106.

Ruby, D., & Kibler, D. (1988). Exploration of case-based problem solving. In J. Kolodner (Ed.), *Proceedings: Cased-Based Reasoning Workshop* (pp. 345–356). San Mateo, CA: Morgan Kaufmann.

Thompson, K., & Langley, P. (1988). Organization and retrieval of composite concepts. In *DARPA* (pp. 329–333). Los Altos, CA: Morgan Kaufmann.