Improving Rule-Based Systems through Case-Based Reasoning[1]

Andrew R. Golding
KSL/Stanford University
701 Welch Road
Palo Alto, CA 94304

Paul S. Rosenbloom
ISI/University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292

Abstract

A novel architecture is presented for combining rule-based and case-based reasoning. The central idea is to apply the rules to a target problem to get a first approximation to the answer; but if the problem is judged to be compellingly similar to a known exception of the rules in any aspect of its behavior, then that aspect is modelled after the exception rather than the rules. The architecture is implemented for the full-scale task of pronouncing surnames. Preliminary results suggest that the system performs almost as well as the best commercial systems. However, of more interest than the absolute performance of the system is the result that this performance was better than what could have been achieved with the rules alone. This illustrates the capacity of the architecture to improve on the rule-based system it starts with. The results also demonstrate a beneficial interaction in the system, in that improving the rules speeds up the case-based component.

1 Introduction

One strategy for improving the performance of a rule-based system is simply to extend its rule set. For many real-world domains, however, this strategy reaches a point of diminishing returns after awhile. The work reported here takes an alternative approach: it performs rule-based reasoning (RBR) with whatever imperfect rules are available, and supplements the rules with case-based reasoning (CBR). This enables the system to tap into a knowledge source that is often more readily available than additional rules; namely, sets of examples from the domain.

The viability of this approach depends on whether adding CBR will actually provide improved coverage of the domain, or merely redundant coverage. Section 3.2 below presents experimental evidence that in fact it provides improved coverage. The reason is that rules and cases have complementary strengths. Rules capture broad trends in the domain, while cases are good at filling in small pockets of exceptions in the rules.

The hybrid strategy is not the only way to incorporate both rules and cases into the system. An alternative is to convert all the cases into rules, or vice versa, and then work in a single representation. Either direction of the conversion has its pitfalls, however. Consider first converting a case into rules. The conversion must preserve the functionality that the case, in its CBR framework, provides. Each case effectively gives rise to a rule every time an analogy is drawn from it, as behind every analogy is an implicit rule. We could therefore represent the case by the set R of all rules that it could ever give rise to. However, R may be huge, as the case could produce subtly different rules for each target problem to which it is analogized. On the other hand, if we leave the case as a case, we only generate rules for target problems that we actually encounter. To economize, we might replace the plethora of rules in R with fewer, more general rules. Induction programs do this by generalizing over multiple cases; this yields across-case economies as well. But no matter how we do it, we are generalizing further than CBR would have generalized from the original case, thus we run the risk of overgeneralization.

Converting in the opposite direction has analogous problems. The conversion from a rule into a set C of cases must preserve the rule's coverage. That is, we must pick C such that for each case covered by the rule, its nearest neighbor is one of the cases in C, according to the similarity metric being used. The only set C that is guaranteed to do this is the full set of cases covered by the rule; but it is likely to be huge, assuming it can be generated at all. If we pare

[1]This research was sponsored by NASA under cooperative agreement number NCC 2-538, and by a Bell Laboratories PhD fellowship to the first author. Computer facilities were partially provided by NIH grant LM05208. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NASA, the US Government, Bell Laboratories, or the National Institute of Health. The authors would like to thank the Speech Technology Group at Bellcore, especially Murray Spiegel, for their valuable assistance in this research. We are also grateful to Pandu Nayak for helpful discussions on the ideas of this paper, and to Ross Quinlan and the AAAI reviewers for useful comments on drafts of this paper.

Finally, it should be mentioned that the system presented here was originally implemented in and shaped by the Soar architecture [Laird *et al.*, 1987]. The Soar influence will not be discussed explicitly in this paper, however.

down C, we run the risk that one of the cases that we delete will be misrepresented. This is because it is unpredictable what the case's nearest neighbor will be in the long run, as further cases are added to the case library. In short, converting either way between rules and cases will tend to produce an inefficient or unreliable representation. The degree to which this happens, and can be tolerated, determines whether it is better to convert or to adopt a hybrid approach. In certain domains, the conversion from cases to rules has been shown feasible by induction programs. But the conversion will leave us with two sets of rules — the original set, and the set derived from cases. The two sets must then be integrated, just as rules and cases must be integrated in a hybrid system. Thus even when it is feasible, induction solves only part of the problem. It leads to an approach of conversion followed by rule integration (see section 4).

The approach taken here has been cast as a general architecture for combining RBR and CBR. The architecture is presented in section 2. The architecture has been implemented for the full-scale task of pronouncing names. Experimental results for this implementation are given in section 3. In section 4, the architecture is compared with alternative methods for combining RBR and CBR. The final section is a conclusion.

2 The Architecture

The central idea of the architecture for combining RBR and CBR is to apply the rules to a target problem to produce a default solution; but if the target problem is judged to be compellingly similar to a known exception of the rules in any aspect of its behavior, then that aspect is modelled after the exception rather than the rules. Problem solving in this architecture is done by applying operators to the problem until it is solved. The central idea above is therefore realized through the following procedure:

```
Until the target problem is solved do:
(a)  Use the rules to select an operator to apply.
(b)  Search for analogies that would contradict
     this choice of operator, stopping if and when
     a compelling analogy is found.[2]
(c)  If a compelling analogy was found, apply the
     operator it suggests, else proceed to apply
     the operator suggested by the rules.
```

Underlying this procedure is the assumption that the rules are decent to begin with. If they are very slow, then the system will suffer when they are applied in step (a). If they are highly inaccurate, then the system will get bogged down overriding them in step (b). In such cases, some alternative architecture is called for, such as one that looks for compelling analogies first, and only consults the rules if none is found.

[2]If there are multiple compelling analogies for different operators, this procedure will only find the first one. This will not result in a wrong answer, unless there is an inconsistency in the case library or the definition of compellingness. It merely will miss the other acceptable answers corresponding to the unchosen analogies.

In the next section, we describe the knowledge needed for the procedure above. We then elaborate on the major aspects of the procedure itself: indexing the cases for use in analogies, proposing the analogies, and deciding when an analogy is compelling.

2.1 Knowledge Sources

The architecture itself is domain-independent; its domain knowledge comes from three external sources: a set of rules, a case library, and a similarity metric. The rules specify an operator to apply in every problem-solving state. The case library is a collection of cases, where a case consists of a problem, its answer, and the chain of operators by which the answer was derived.[3] The similarity metric gauges the similarity between two problems for purposes of applying a particular operator. It will be discussed further in section 2.3.

2.2 Indexing

The role of CBR in the architecture is to improve the performance of the rules. It follows that the only cases from which useful analogies can be drawn are the exceptions, i.e., the cases that violate the rules. Cases that confirm the rules do not lead to any new behaviors. Moreover, the only time an exception is useful is when the rule that it violates actually fires. At that point, it becomes relevant to ask whether the exception should override the rule. These considerations lead to the indexing scheme of storing each case as a *negative exemplar* of the rules that it violates. To determine which rules these are, we basically apply RBR to the case as if it were a new problem. If, in the process, a rule R says that a certain operator should apply, but the case library specifies that in fact some other operator was applied, then the case violates rule R. It turns out to be useful to store each case also as a *positive exemplar* of the rules that it confirms — this will help later in judging compellingness (see section 2.4). We call this scheme *prediction-based indexing* (PBI), because effectively, an exemplar is indexed by the features that the rules looked at in order to to predict which operator to apply. This is like explanation-based indexing [Barletta and Mark, 1988], except that there the rules are used to explain an observed outcome, rather than to make their own prediction of the outcome.

Example For ease of exposition, we will illustrate the architecture not for name pronunciation, but for a toy version of a problem in auto insurance: to assess the risk of insuring a new client. Problem solving consists of applying just one operator, either `high` or `low`,

[3]Actually, the chain of operators need not be specified; the architecture can infer it from the problem/answer pair by a process of *rational reconstruction* [Golding, 1991].

```
If occ(C) = student then low          ; 'Student' rule
elseif sex(C) = M and
    age(C) < 30 then high             ; 'Young driver' rule
elseif age(C) ≥ 65 then high          ; 'Old driver' rule
else low                              ; 'Default' rule
```

Figure 1: The rules in the toy auto-insurance example. C stands for a client.

which asserts the level of risk of the client. The full set of rules is shown in Figure 1. A client is represented as a feature vector; Figure 2 gives some examples. The case library contains 23 cases. Each case specifies a client and the operator applied to that client, either `high` or `low`. The cases are derived (conceptually) from the insurance history of past clients.

To illustrate PBI, we consider the first client in the case library, Johnson. Suppose that he is listed as having had the `high` operator applied. The first step of PBI is to see what the rules would have predicted. They predict `low` by the 'student' rule. Since this disagrees with the case library, Johnson is stored as a negative exemplar of the 'student' rule. As a second example, consider the client Davis, whom we will suppose is listed as `low` in the case library. Again the 'student' rule applies, but this time its prediction agrees with the case library. So Davis is listed as a positive exemplar of the 'student' rule.

2.3 Proposing Analogies

Proposing analogies is done by applying the similarity metric. The metric takes three arguments: the source and target problems, and the operator to be transferred from source to target. The operator establishes a context for comparing the problems. Given these three arguments, the metric returns two values: a numerical rating of the similarity (the *similarity score*), and the implicit rule behind the analogy (the *arule*). The left-hand side of the arule gives the features that were judged by the metric to be shared by the two problems, and the right-hand side gives the operator-to-be-transferred. The arule will be used for judging whether the analogy is compelling (see section 2.4).

Example Continuing with the insurance example, suppose the system is asked to evaluate the risk of client Smith (see Figure 2). It starts by applying the rules to Smith. The 'student' rule matches, suggesting the `low` operator. Before accepting this conclusion, the system checks for analogies from negative exemplars of the rule. As we saw earlier (section 2.2), Johnson is one such negative exemplar. Application of the similarity metric for this domain to Johnson and Smith (with respect to the `high` operator) yields the arule:

If addr1(C) = Sigma Chi **and**
addr2(C) = Stanford, CA **and** sex(C) = M
and age(C) < 30 **and** occ(C) = student
then `high`.

	Target	**Case #1**	**Case #6**
Name	Smith	Johnson	Davis
Addr1	Sigma Chi	Sigma Chi	Toyon Hall
Addr2	Stanford, CA	Stanford, CA	Stanford, CA
Sex	M	M	F
Age	21	19	22
Occ	student	student	student
Make	Chevrolet	BMW	Toyota
Value	2,500	30,000	3,000

Figure 2: Selected clients in the insurance example.

This arule expresses the features shared by Johnson and Smith, according to the metric. The metric is very simplistic in this toy domain. It compares corresponding text fields of the two clients via literal comparison. For numeric fields, it checks whether the two numbers fall within the same interval of a predefined set of intervals. It assigns similarity scores by counting the conditions in the arule; here the score is 5.

2.4 Deciding Compellingness

Rather than accepting an analogy purely on the basis of its similarity score, the system subjects it to inductive verification. This entails testing out the arule of the analogy on all relevant exemplars — both negative and positive — in the case library. The test returns two results: the arule's accuracy, that is, the proportion of cases it got right; and the significance of the accuracy rating, which is 1 minus the probability of getting that high an accuracy merely by chance. The analogy is then said to be compelling iff (1) its similarity score is high enough, (2) its accuracy is high enough, and (3) either its accuracy rating has a high enough significance, or its similarity score is extremely high. The latter disjunct is an escape clause to accept analogies between overwhelmingly similar problems, even if there are not enough data for a significant accuracy reading. All "high enough" clauses above are implemented via comparison with thresholds. The thresholds are set by a learning procedure that generates training analogies for itself from the case library [Golding, 1991].

Example Consider again the analogy from Johnson to Smith. Should it be judged compelling? To decide, the system first runs an inductive verification. It turns out that the arule applies to four clients in the database: Johnson and three others. Three of them are listed as `high` risk, one as `low`. This gives an accuracy of 3/4. The significance of this accuracy rating works out to be 0.648. Also, as mentioned earlier, the similarity score of the analogy is 5. The thresholds in this domain have been set to 3 for the similarity score, 0.75 for accuracy, and 0.50 for significance.[4] Thus the anal-

[4]These thresholds were set by hand rather than by the usual learning procedure, because the toy domain has too few cases to apply the learning procedure meaningfully.

ogy is deemed compelling, although it was marginal in the accuracy department. The upshot is that Smith is assessed as high risk, by analogy to a similar high-risk student from the same fraternity.

3 Experimental Results

The architecture described here was developed in the context of Anapron, a system for pronouncing surnames. In particular, the architecture was applied to two subtasks of pronunciation: transcription and stress assignment. Transcription converts a spelling into a string of phonetic segments. Stress assignment places a level of emphasis on each syllable. Although there has been little work on pronunciation within the CBR community — with a few notable exceptions [Lehnert, 1987; Stanfill, 1987] — the domain was selected for the present research for several reasons. First, both rules and cases are already available; rules have been developed in previous pronunciation efforts [Hunnicutt, 1976], and case libraries can be derived from pronouncing dictionaries of names. This makes the domain amenable to the hybrid rule/case approach. Second, pronouncing *names* in particular is an open problem [Klatt, 1987], due to the unique etymology and morphology of names. To give an idea of the size of the system, there are 619 transcription rules and 29 stress rules, covering five major languages. There are 5000 cases in the system, derived from a name dictionary of the same size. Below we give results on the overall level of performance of the system, and on the contribution of RBR and CBR to this performance.

3.1 Overall Performance[5]

To establish the initial credibility of the architecture for the pronunciation domain, we include here the results of a pilot study comparing Anapron with six other name-pronunciation systems: three state-of-the-art commercial systems (from Bell Labs, Bellcore, and DEC), one machine-learning system (NETtalk [Sejnowski and Rosenberg, 1987], trained on Anapron's name dictionary), and two humans. Each system was run on a test set of 400 names, and the acceptability of its pronunciations was measured. This had to be done by taking a poll of public opinion, as there is no objective standard for surname pronunciations. To hide the identities of the systems in the poll, the order of systems was randomized for each test name, and all pronunciations were read by the DECtalk speech synthesizer. The pilot study was conducted on just one

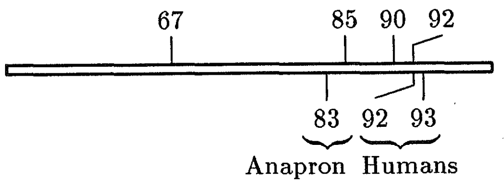

Figure 3: Results of pilot study comparing seven name-pronunciation systems. Each line marks the percentage of acceptable pronunciations for one system. The scale goes from 50% to 100%.

test subject; at this point, the exact numbers in the results should not be taken too seriously.

The test set for this experiment was drawn from the Donnelly corpus, a database of over 1.5 million surnames in the US. The names in Donnelly range from extremely common (Smith, which occurs in 676,080 households) to extremely rare (Chavriacouty, which occurs in 1 household). The test set contains 100 randomly selected names from each of four points along this spectrum: names that occurred in about 2048 households, 256 households, 32 households, and 1 household. Rare names are known to be harder to pronounce than common ones. The test set therefore represents a fairly challenging cross-section of Donnelly.

Results The results of the pilot study are shown in Figure 3. The identities of most systems have been omitted due to the preliminary nature of the results. The initial indications are that Anapron performs near the level of the best commercial systems, which is barely short of human performance. One reason that the commercial systems may outperform Anapron is simply that they have better rules. Presumably, giving these same rules to Anapron would help its performance; moreover, the point of Anapron is that it can then achieve further gains by leveraging off CBR.

3.2 The Contribution of RBR and CBR

An experiment was run on Anapron to evaluate the effect of combining RBR and CBR in practice. The results can be taken as one example of how the architecture behaves when instantiated for a task. The experiment involved independently varying the strength of the rules and of the case library, and observing how system performance was affected on a particular test set. The rules were set to four different strengths: 0, 1/3, 2/3, and 1. Strength 1 means that all rules were retained in the system; 0 means that all rules were deleted, except *default* rules.[6] As strength decreases

[5]The authors gratefully acknowledge the assistance on this experiment of the following people: Cecil Coker at Bell Labs, Murray Spiegel at Bellcore, and Tony Vitale at DEC, for supplying data from their systems; Tom Dietterich, for providing the non-copyrighted portion of NETtalk; John Laird, for providing a fast machine for training NETtalk; Mark Liberman, for making the test set of names available; and Connie Burton, for providing access to DECtalk.

[6]The rules are arranged in a partial order by specificity; more specialized rules take precedence over more general ones. A default rule is one that is maximally general. Such a rule must not be deleted, otherwise the system will no longer be guaranteed to produce an answer for every problem. Default rules constitute 137 of the 619 transcription rules and 16 of the 29 stress rules.

Rule strength	Case strength 0	1000	2000	3000	4000	5000
0	19	27	32	34	35	36
1/3	33	39	43	44	46	47
2/3	46	54	56	56	57	59
1	56	65	65	67	67	68

Table 1: System accuracy results. Each value is the percentage of names in the test set for which the system produced an acceptable pronunciation.

Rule strength	Case strength 0	1000	2000	3000	4000	5000
0	0.9	3.0	4.5	6.3	8.2	10.2
1/3	1.1	3.0	4.4	5.9	7.6	9.3
2/3	1.2	3.1	4.4	5.8	7.2	8.5
1	1.3	3.0	4.1	5.2	6.3	7.2

Table 2: System run-time results. Each value is the average time, in seconds, for the system to pronounce a name in the test set.

from 1 to 0, we delete proportionately more rules. Each weakening deletes a random subset of the non-default rules in the previous rule set. As for the case library, it was set to six different strengths: 0, 1000, 2000, 3000, 4000, and 5000. The strength is just the number of names included in the case library. Each weakening deletes an arbitrary subset of the previous case library.

System performance was measured by two parameters: accuracy and run time. Accuracy is the percentage of names in the test set for which the system produced an acceptable pronunciation. The same 400-name test set was used as in the previous experiment, except that the names were chosen to be disjoint from the case library. This time the decisions of acceptability were made by a single, harsh human judge (the first author).[7] All judgements were cached and reused if a pronunciation recurred, to help enforce consistency. The other parameter of system performance, run time, is just the average time, in seconds, for the system to pronounce a name in the test set. The data are for the system running in CommonLisp on a Texas Instruments Microexplorer with 8M physical memory, and are inclusive of garbage collection and paging.

Accuracy Results System accuracy, for each combination of rule and case strength, is shown in Table 1. The important result is that accuracy improves monotonically as rule or case strength increases. The total improvement in accuracy due to adding rules is between 32% and 38% of the test set (depending on case strength). For cases it is between 12% and 17% (depending on rule strength). This shows that by combining rules and cases, the system achieves a higher accuracy than it could with either one alone — both are essential to the accuracy of the combined system. This suggests dual views of the architecture: as a means of improving rule-based systems through CBR, or improving case-based systems through RBR.

Run-time Results Table 2 gives the results on run time. The interesting point here is that when the case library is large, adding rules to the system actually decreases run time. For example, with the case library at size 5000, increasing the rules from strength 0 to 1 lowers run time from 10.2 to 7.2 seconds per name. The basic reason is that adding rules to the system improves the overall accuracy of the rules, barring sociopathic effects. When the rules are more accurate, they will have fewer exceptions. This translates into fewer negative exemplars, and thus fewer opportunities to draw analogies. The forgone analogies result in a corresponding savings in run time. In short, adding rules to the system speeds up the CBR component. This shows that RBR and CBR do not merely coexist in the system, they interact beneficially.

[7]The resulting scores are not directly comparable to those in section 3.1, primarily because the judge there applied native-speaker intuitions to the spoken pronunciations, whereas the first author applied more formal notions of acceptability to the written transcriptions.

4 Related Work

A number of other methods have been proposed in the literature for combining RBR and CBR. They fall into two basic classes, according to whether their rules and cases are independent, or whether one was derived from the other. The primary motivation for the former class of systems is to maximize accuracy by exploiting multiple knowledge sources. For the latter class of systems, it is to express their knowledge in whatever form will make problem solving most efficient.

The systems whose rules and cases were derived from each other can be further classified according to which knowledge source was derived from which. Most CBR systems that include a rule component [Koton, 1988; Hammond and Hurwitz, 1988, etc.] have cases that are derived from their rules. The cases are records of how the rules were applied to particular examples encountered previously. By reasoning from cases, the systems bypass the potentially lengthy process of solving a new problem from scratch via the rules. The systems whose rules are derived from their cases extract the rules by some generalization procedure. The systems must still keep the cases around, because their rules do not encode all of the knowledge in the cases. The rules in these systems can serve various purposes, such as enabling a more compact representation of the data [Quinlan and Rivest, 1989], or providing more efficient access to the cases [Allen and Langley, 1990].

Systems utilizing independent rules and cases are much closer in spirit to the system described here. Again the systems fall into two groups. In the first group [Rissland and Skalak, 1989; Branting, 1989], the focus is on deciding how and when it is appropriate to invoke RBR and CBR. For example, CABARET [Rissland and Skalak, 1989] uses heuristics for this purpose. These systems do not try to reconcile conflicts between RBR and CBR, they merely report all of the evidence.

In the second group of systems, the focus is on reconciling the conclusions of RBR and CBR. Anapron falls into this group, and so does MARS [Dutta and Bonissone, 1990]. MARS represents cases not as cases per se, but as rules derived from the cases. It acquires these rules from written documents via natural-language processing. The documents in MARS's domain of mergers and acquisitions are the rulings of the judges who decided each case. Once everything is represented as rules, MARS is able to aggregate the evidence from multiple rules using possibilistic reasoning. This requires that each rule specify a level of necessity and sufficiency with which its conclusion is implied. Thus MARS takes the approach mentioned earlier (see section 1) of converting cases to rules, and then integrating the derived and original rules.

The fundamental difference between MARS and Anapron is that MARS specifies the knowledge for drawing analogies on a per-case basis, whereas Anapron specifies it all at once in a more general form. That is, in MARS, each case is written as a rule that says which of its features must match the target problem; it also gives necessity/sufficiency values that specify its strength for purposes of aggregation. In Anapron, the similarity metric gives the equivalent knowledge for matching cases and evaluating their strength. This indicates that the two systems are appropriate in different situations. When it is practical to do the knowledge engineering of cases that MARS requires, MARS is appropriate. When it is practical to specify a similarity metric, Anapron is appropriate.

5 Conclusion

A general architecture was presented for improving the performance of rule-based systems through CBR. The motivation for turning to CBR was that cases are often easier to obtain than additional rules. The architecture was implemented for the full-scale problem of name pronunciation. Preliminary results indicate that the system performs near the level of the best commercial systems. However, this only reflects how the system does with its current rules and cases. The more significant finding was that the system could not have achieved this level of accuracy with its rules alone. This illustrates the capacity of the architecture to improve on the rule-based system it starts with. The results also showed that RBR and CBR interact beneficially in the system, in that improving the rules speeds up CBR. Finally, the architecture fills a new niche among rule/case hybrids — it is an accuracy-improving system; it focuses on reconciling the conclusions of RBR and CBR; and it requires weak domain knowledge in the form of a similarity metric, instead of complex knowledge engineering of the case library.

Directions for future work include improving the system for pronunciation; applying the architecture to other domains; allowing the system to save its arules; and generating similarity metrics automatically.

References

John A. Allen and Pat Langley. A unified framework for planning and learning. In *Proc. of Work. on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, 1990. Morgan Kaufmann.

Ralph Barletta and William Mark. Explanation-based indexing of cases. In *Proceedings of the CBR Workshop*, Clearwater Beach, 1988.

L. Karl Branting. Integrating generalizations with exemplar-based reasoning. In *Proceedings of the CBR Workshop*, Pensacola Beach, 1989.

Soumitra Dutta and Piero Bonissone. Integrating case based and rule based reasoning: The possibilistic connection. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, July 1990.

Andrew R. Golding. *Pronouncing Names by a Combination of Case-Based and Rule-Based Reasoning.* PhD thesis, Stanford University, 1991. Forthcoming.

Kristian J. Hammond and Neil Hurwitz. Extracting diagnostic features from explanations. In *Proc. of CBR Workshop*, Clearwater Beach, 1988.

Sharon Hunnicutt. Phonological rules for a text-to-speech system. *American Journal of Computational Linguistics*, 1976. Microfiche 57.

Dennis H. Klatt. Review of text-to-speech conversion for English. *J. Acoust. Soc. Am.*, 82(3), 1987.

Phyllis Koton. Reasoning about evidence in causal explanations. In *Proc. of AAAI-88*, St. Paul, 1988.

John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33, 1987.

Wendy G. Lehnert. Case-based problem solving with a large knowledge base of learned cases. In *Proceedings of AAAI-87*, Seattle, 1987.

J. Ross Quinlan and Ronald L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80, 1989.

Edwina L. Rissland and David B. Skalak. Combining case-based and rule-based reasoning: A heuristic approach. In *Proc. of IJCAI-89*, Detroit, 1989.

Terrence J. Sejnowski and Charles R. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1, 1987.

Craig W. Stanfill. Memory-based reasoning applied to English pronunciation. In *Proceedings of AAAI-87*, Seattle, 1987.