

**AGASSISTANT: AN EXPERIMENTAL EXPERT
SYSTEM BUILDER FOR AGRICULTURAL
APPLICATIONS**

by

B. Katz
T. W. Fermanian
R. S. Michalski

ISG Report 87-16, UIUCDCS-F-87-978, Dept. of Computer Science,
University of Illinois, Oct.. 1987.

AgAssistant: An Experimental Expert System Builder for Agricultural Applications

by

**Bruce Katz
Thomas W. Fermanian
Ryszard S. Michalski**

*Department of Computer Science
and
Department of Horticulture
University of Illinois at Urbana-Champaign
Urbana, Illinois*

FILE NO.: UIUCDCS-F-87-978

ISG Report 87 - 16

October, 1987

This research was supported in part by the International Intelligent Systems, Inc.; the University of Illinois Research Board; and Project No. ILLU-65-0357, of the Agricultural Experiment Station, College of Agriculture, University of Illinois at Urbana-Champaign.

AgAssistant: An Experimental Expert System Builder

Abstract

AgAssistant is a comprehensive expert system builder for IBM PC and compatible computers in the area of agriculture. The inferencing mechanism was specifically designed to combine levels of uncertainty commonly found in agricultural domains. It is both an enhancement and an extension of an earlier expert system for the IBM PC, PLANT/ds, which was concerned with the diagnosis of soybean diseases common in Illinois. Unlike PLANT/ds, in which all modifications of the system take place on a VAX minicomputer, AgAssistant provides a set of tools for system modification and development directly on the PC. The work presented here is also based to a large extent on the ADVISE Meta-Expert System.

AgAssistant is also more than an expert system builder. It provides a set of tools for the development of a Knowledge base by example of expert knowledge (learning by example). These learning tools may be applied to other agricultural problems where the classification or grouping of non-nummaric data is useful.

Among the many novel features incorporated into AgAssistant are:

- Multiple means of creating and refining knowledge.

AgAssistant can use rules developed directly from experts or acquired through inductive inference in the same format .

- Probabilistic inference can be handled.

Of great use in agriculture, where the vagaries of nature can make identification or diagnosis a probabilistic matter.

- The system is PC-based.

This allows wide dissemination of the program to farmers and others in need who are unlikely to have access to larger systems.

- Menu-driven screens.

The novice user can quickly come up to speed in building experts systems.

Table of Contents

1. Introduction	1
1.1 An Overview of the program	1
1.2 Knowledge Representation in AgAssistant	2
1.3 Relevance to Agriculture	4
2. The Advisory System	5
2.1 System Representation	5
2.2 Inference mechanism	8
2.2.1 Individual rule evaluation	8
2.2.2 Evaluation of rules in a hierarchical rule base	10
2.3 Control mechanism	10
2.3.1 Utility control scheme	10
2.3.2 Backtracking control scheme	11
3. Knowledge Acquisition Facilities	13
3.1 Direct Editing of Rules	14
3.2 Rule Learning	14
3.2.1 Learning Rules from Examples	15
3.2.2 Improving Rules with Examples	15
3.2.3 Rule Optimization	16
3.3 Rule Compilation	17
4. An Exemplary System WEEDER	19
4.1 Development of WEEDER	19
4.2 Validation of WEEDER	22
5. Conclusions	25
6. AgAssistant User's Guide	26
6.1 The Main Menu	27
6.1.1 Getting Advice from an Expert System	28
6.1.2 Getting Advice from a particular system	29
6.1.3 Options while answering a question	30
6.2 The Windows	33
6.2.1 The Plan Window	33

AgAssistant: An Experimental Expert System Builder

6.2.2 The Answers Window	34
6.2.3 The Confirmation Window	34
6.2.4 The end of an advisory session	35
6.3 Develop a New or Improve an Old Expert System	36
6.3.1 Accessing Rules	37
6.3.2 Creating Rules	38
6.3.3 Accessing or Creating Examples	45
6.3.4 Accessing or Creating Variables	48
6.3.5 Accessing or Creating System Description	51
6.3.6 Copying	52
6.3.7 Deleting	53
6.3.8 Printing	53
6.4 Variable Types	53
6.4.1 Nominal variables	53
6.4.2 Linear variables	53
6.4.3 Integer variables	53
6.4.4 Numeric variables	54
6.4.5 Structured variables	54
6.5 Rule Compiler Error Messages	55
7. Rules for WEEDER	57
7.1 Modified WEEDER Rules	62
8. References	63

List of Figures

Figure 1. An Overview of AgAssistant	2
Figure 2. A hypothetical rule in the knowledge base of an expert system	3
Figure 3. An illustrative rule base	5
Figure 4. Variable Definitions for illustrative system	6
Figure 5. An example of a compiled expert system	7
Figure 6. General rule format	8
Figure 7. Utility control scheme	11
Figure 8. Backtracking Control Scheme	12
Figure 9. Knowledge Acquisition in AgAssistant	13
Figure 10. Rules before and after optimization	17
Figure 11. Grammar for rules	18
Figure 12. Variable names, types, and values for WEEDER	20
Figure 13. A rule for identifying Stinkgrass	21
Figure 14. Do not apply conditions in WEEDER	21
Figure 15. An Overview of the AgAssistant Menu Structure	26
Figure 16. The AgAssistant Main Menu	27
Figure 17. Getting advice from an Expert System	28
Figure 18. Responding to a question requiring a user to select an answer	30
Figure 19. Responding to a questions requiring a numeric answer	31
Figure 20. Screen for viewing confidence levels	32
Figure 21. Typical entries in the plan window.	34
Figure 22. The final screen of the advisory session	35
Figure 23. Developing an Expert System	37
Figure 24. Accessing Rules	38
Figure 25. Rule Compilation	39
Figure 26. Creating rules	39
Figure 27. Learning Rules from Examples	40
Figure 28. Screen displayed while learning rules	41

AgAssistant: An Experimental Expert System Builder

Figure 29. Improving Rules with Examples	43
Figure 30. Editing learning parameters	44
Figure 31. Editing examples	46
Figure 32. Editing variables	49
Figure 33. Editing System Description	51
Figure 34. Structure for the variable shape	54

1. Introduction

AgAssistant is a comprehensive expert system builder for personal computers in the domain of agriculture. While it may be used to construct expert systems in any domain, its inference system was designed specifically to handle the uncertainty found in many agricultural domains. It is an extension of a conceptual predecessor, Plant/ds, an earlier expert system for the IBM PC which was concerned with the diagnosis of soybean diseases common in Illinois and had the capacity to learn its rules through inductive learning by example [Michalski et al., 1982]. Unlike Plant/ds, in which one interacts with the VAX minicomputer to build and refine the knowledge base, one need not leave the PC environment, either in creating an expert system, or in getting advice from a system that already exists. The work here is also based to a large extent on the ADVISE Meta-Expert System [Michalski and Baskin, 1983]

Among the many important features of AgAssistant are:

- Multiple means of creating and refining knowledge.

AgAssistant can receive rules directly or acquire them through inductive inference.

- Probabilistic inference is employed for handling uncertainty of data and rules.

It is of great use in agriculture, where the vagaries of nature make identification or diagnosis a probabilistic matter.

- Implemented on a personal computer.

This allows wide dissemination of the program to farmers and others in need who are unlikely to have access to larger systems.

- Menu-driven screens.

The novice user can quickly come up to speed in building experts systems.

1.1 An Overview of the program

AgAssistant consists of a set of modules accessible through menus-driven screens (see Figure 1). The Advisory Module takes as its input a compiled expert system, which it uses to create questions for the user, as well as give advice on the basis of the answers to these questions. It is described in more detail in chapter 2. The compiled system is created by the compiler, which, in addition to parsing rules for correct syntax, creates a more compact version of the system for faster execution. Rules may be created by hand, or created through induction on examples. The inductive module calls the program NEWGEM [Reinke, 1984] to operate on examples, and variable definitions to produce rules. Additionally, this module may start with existing rules and improve them with ex-

AgAssistant: An Experimental Expert System Builder

amples (incremental learning), or optimize them according to differing criteria. The methods for building, refining, and compiling an expert system are described in chapter 3. The User's Guide, chapter 6 of this report, describes how one would actually go about using AgAssistant.

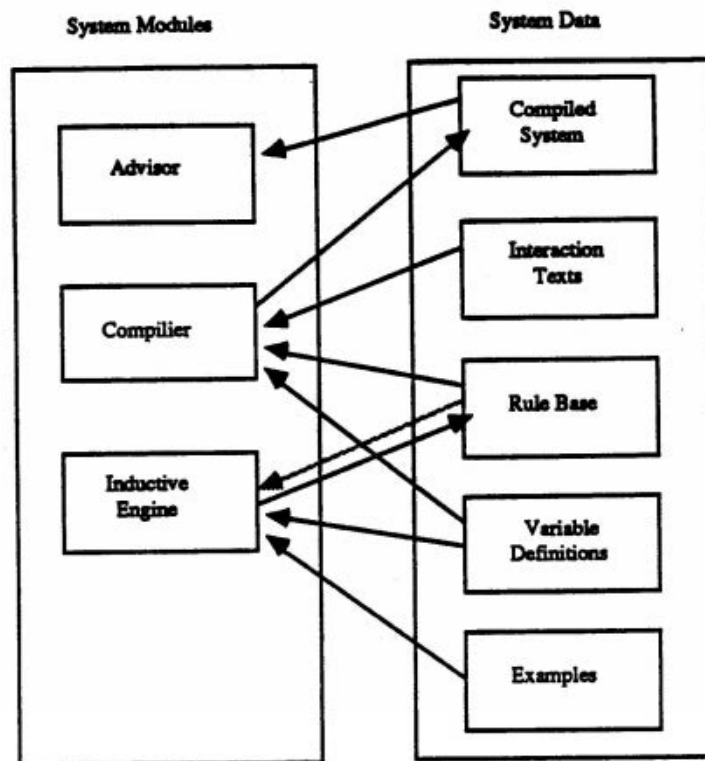


Figure 1. An Overview of AgAssistant

1.2 Knowledge Representation in AgAssistant

Knowledge is represented in AgAssistant primarily in the form of VL1 rules [Michalski, 1975]. Variables in these rules may be nominal, linear, integer, structured, or numeric. The exact syntax and constraints on the rules appears in section 4.3, while the various variable types and their representations is described fully in chapter 6. Here we will simply give the reader a feel for the way knowledge is represented so that he or she may understand better the following chapters. A hypothetical rule that incorporates all of the allowed variable types appears in Figure 2.

Crop should_be harvested if:	cl
1. Crop_length*crop_width > 12 inches,	50
2. Crop_shape is oblong,	20
3. Temperature is 65 to 75,	15
4. Soil_moisture is medium to high,	10
5. Sky is sunny.	5
OR	
1. Month is October,	75
2. Weather is fair.	25

Figure 2. A hypothetical rule in the knowledge base of an expert system

The above rule contains two complexes (conjunctive set of conditions), the first of which consists of five conditions (or selectors), while the second complex contains two selectors. Each selector is followed by a weight or confidence level (cl), which indicates the relative importance of the selector as a condition for the decision concept. For example, if the only fact which is known is *the crop shape is oblong* then the expert is 50 % confident that the crop should be harvested. The method of combining these weights if more than one fact is known is explained in chapter 2. The action for this rule, namely that the crop should be harvested, depends on whether those conditions are satisfied. The degree to which the set of conditions must be satisfied can be set in the system threshold (Chapter 2.). Unlike other inferencing mechanisms, this system can support a decision with only an approximate match of the evidence to the stated conditions.

The first condition of the first complex will be fulfilled if the arithmetic expression is true, that is, if the product of the *crop_length* and the *crop_width* exceeds 12.5 inches. The second condition will be satisfied if the shape of the crop is oblong. Since this is a structured variable, values of the variable are arranged in a hierarchy; thus this condition will be true if *crop_shape* takes the value of oblong, or any child of oblong. The third selector will be true if the temperature is between 65° and 75°. The fourth selector will be fulfilled if *soil_moisture* is in the range of medium to high; there may be values in between medium and high (such as medium-high) which are implicitly included in this selector. The last condition will be satisfied only if the nominal variable *sky* receives the value sunny, out of a possible set of values including raining, cloudy, etc. The second complex of the rule, following the 'OR' consists of two selectors and says more or less that if you haven't brought in the crop by October, you should do so now if the weather is fair.

AgAssistant: An Experimental Expert System Builder

A knowledge base for an expert system consists of a set of such rules. Rules may also be structured in a hierarchical fashion, i.e., the condition of one rule may be the action of another rule. The process of building a knowledge base is described in more detail in the User's Guide. The reader may also refer to Chapter 4 which outlines the exemplary expert system WEEDER to see a complete expert system.

1.3 Relevance to Agriculture

Many underlying principles of agricultural sciences can be experimentally measured in field experiments. The results of these experiments, however, show a normal abundance of variation in measured responses which is expected in nature. This natural variation has made the development of realistic models of agricultural systems difficult. Many assumptions are required for even the most simple crop or environmental model. Expert systems technology, for the first time, will offer a technique for working with fuzzy or uncertain knowledge.

Agricultural scientists often provide advice to agricultural managers on the basis of an evaluation of their incomplete knowledge and experience. This closely parallels the process of an expert system. Due to natural variability, agricultural knowledge bases are unstable and require continual modification to reflect current conditions or knowledge. Advisory systems which require the intervention of mainframe computers or centralized systems development cannot keep up with the rapid changes necessary. AgAssistant represents an expert system development environment which can be easily modified in the field. It can therefore truly reflect any changes seen in the natural responses of the model in question.

While many agricultural production processes are inherently complex, a subgroup of production processes can adequately be described and converted to an appropriate knowledge base for use with AgAssistant. An example of these are pest or crop identification or pest damage diagnosis systems. In addition, simple designing of agricultural production systems would be appropriate domains for AgAssistant. Oftentimes agricultural data is of a subjective, qualitative nature which might be more rapidly and thoroughly analyzed through symbolic processing techniques. AgAssistant represents a new technology in the form of a tool to assist agricultural scientists and managers to better interpret the observed phenomenon.

2. The Advisory System

At the heart of the AgAssistant advisory system is a flexible inference engine that runs on a compiled set of rules. The exact form of this file is described in detail in section 1 of this chapter. Section 2 explains how individual rules are evaluated and how uncertainties are propagated through a hierarchical knowledge base. Section 3 explains the control structure of the system, i.e., the method by which the advisory system generates questions. This chapter explains the technical details upon which the advisory system is based. The reader should refer to the User's Guide for information on getting advice from an actual expert system.

2.1. System Representation

The rule parser takes a set of rules and a set of variable definitions and produces a file containing the compiled version of the expert system. This file consists of a cross-referenced version of the original system. Figure 3 shows a section of a rule base for an expert system, while Figure 4 illustrates the definitions of the variables involved.

Crop should be harvested if:	cl
1. Crop is ripe,	60
2. Weather is good.	40
Crop is ripe if:	
1. Crop_color is yellow or green,	50
2. (Crop_width * Crop_length) > 12.	50
Weather is good if:	
1. Sky is sunny,	50
2. Soilmoisture is low,	50
3. Windstrength is very_mild to medium	25

Figure 3. An illustrative rule base

The rules in the above system represent a section of a hypothetical expert system for crop management. Undoubtedly, an actual system, would contain many additional rules, (e.g. rules for fertilization, plowing, etc.) and each rule would be of greater complexity; this rule base is meant solely for illustrative purposes. The rules indicates that the crop should be harvested if the weather is good and the crop is ripe. Each of these conditions are in turn based on further conditions as indicated in the first two rules. The variables and their respective types and domains are shown in Figure 4. The compiled system appears in Figure 5.

AgAssistant: An Experimental Expert System Builder

VARIABLE	Crop_length	Crop_width	Weather	Soilmoisture	Crop_color	Windstrength	Crop	Sky
TYPE	numeric	numeric	linear	linear	nominal	linear	nominal	nominal
Value 1	0-50	0-10	bad	low	yellow	very_mild	ripe	sunny
Value 2			fair	medium	green	mild	unripe	cloudy
Value 3			good	high	brown	medium		rainy
Value 4					black	mediumstrong		
Value 5						strong		
Value 6								
Value 7								
Value 8								

Figure 4. Variable Definitions for illustrative system

Before explaining the desirability of converting the original system into its compiled form, the format of the file in Figure 5 will be explicated. The file consists first of a list of variables and information relevant to them. For example, the variable *Crop_color* is the fifth variable in the list (after *Soilmoisture*). Following the variable name is the relation used with this variable, and then the values associated with the variable obtained from the variable table. Following each value is a list of tuples, each containing four elements, viz. rule number, rule complex, selector within the complex, and a weight. For example, if *Crop_color* takes the value yellow, then selector 1 of complex 1 of rule 2 (rules are given numbers in order of appearance within the rule base) will be updated by 50% (the weights are represented as percentages with an additional digit of significance to reduce the possibility of roundoff error). Values may have more than one tuple following them if the variable value pair appears in more than one complex. Variables that are numeric and which appear in arithmetic expressions, are indicated as such in the line of #'s separating the variables.

Following the variables, appear two lines of #'s, after which appear the rule actions. Each action has three lines of information. The first line is simply the action verbatim, as it appears in the rule base. The second line, possibly blank, contains the name of a text file (e. s., *harvest.txt* for the rule "Crop should_be harvested") invoked if the rule associated with the action receives the highest confirmation at the completion of the advisory session. The third line consists of a number that indicates which variable, if any, is identical with the action, followed by a list of four-tuples, with the same ordering as described above, which list the locations of the action as conditions in other rules. For example, this line for the action 'Crop is ripe' begins with an 8 to indicate that this action variable, viz., 'crop', is also the eighth variable in the variable list at the beginning of the file. Additional information indicates that the action also appears in rule 1, in the first selector of the first complex.

AgAssistant: An Experimental Expert System Builder

numeric		Sky	
Crop_length		is	
0-50		cloudy	
#####		rainy	
numeric		sunny	1 1 1 500
Crop_width		#####	
0-10		Crop	
#####		is	
Weather		ripe	3 1 1 600
is		#####	
bad		#####	
fair		Crop should be harvested	
good	3 1 2 399	harvest.txt	
#####		3 8	
Soilmoisture		Crop is ripe	
is		8 1 1 1	
low	1 1 2 250	1 2 5	
medium		Weather is good	
high		3 1 1 2	
#####		4 6 7	
Crop_color		#####	
is		1 2 3 4 5 6 7 8	
yellow	2 1 1 500	#####	
green	2 1 1 500	\$\$ Arithmetic expressions	
brown		\$\$ (Crop_width * Crop_length) > 12	2 1 2 500
black		#####	
#####		Weather	What is the weather like?
Windstrength		Soilmoisture	To what extent is the soil saturated?
is		Crop_color	What color is the crop?
very_mild	1 1 3 250	Windstrength	How strong is the wind?
mild	1 1 3 250	Sky	What is the appearance of the sky?
medium	1 1 3 250	Crop	Is the crop ripe yet?
mediumstrong		Crop_length	What is the length of the crop?
strong		Crop_width	What is the width of the crop?
#####			

Figure 5. An example of a compiled expert system

In the fourth and last line the order which variables should be asked is presented (see section 2.3 for an explanation of this ordering). Following this is a list of arithmetic expressions found throughout the rule base, with the appropriate tuple list trailing each. Finally, variables and questions that will be asked during the advisory session when the system wishes to know the value of a variable are listed.

One may contend that the information contained in the compiled system, is, with a few minor additions, just a rehashing of the system in its original form. While this is correct, there is an important reason for representing the system in such a way. Consider what needs to be done to update a rule after a new value is associated with the

AgAssistant: An Experimental Expert System Builder

variable. Suppose the system consists of n rules each with an average of s selectors. One must then search the entire rule space for the appearance of the variable-value pair, or perform ns searches. Additionally, in the worst possible case, one must search for the actions appearing as selectors in other rules resulting in sn^2 matches, or ns matches for each of n rules. While both of these figures are polynomial quantities, they will nevertheless prove prohibitively large for a PC system if n is large. Thus the appearance of all values, as well as the location of all actions as conditions in other rules, are cross-referenced beforehand in the compiled file, resulting in almost no search. The exact method of updating rule confidence levels, once this information is provided, is examined in the following section.

2.2. Inference mechanism

Rules have the general format shown in figure 6. Since the conditions in rules are annotated by weights to represent strength of evidence in favor of the decision, it is not sufficient to simply invoke the standard laws of deductive inference in evaluating confidence levels of the rules. Section 2.2.1 describes the method for evaluating individual rules, while Section 2.2.2 shows how rules are evaluated in a hierarchical system.

Action is xxxxx if:	confidence level	
1. variable1 is value1,	60	} Complex1
2. variable2 is value2 or value3,	40	
3. variable3 is value4 to value6.	20	
OR		
1. variable4 is value7,	50	} Complex2
2. variable5 is value8.	50	

Figure 6. General rule format

2.2.1 Individual rule evaluation

The complex of a given rule is transparent to evaluate given the form of the compiled system described in Section 2.1. A sum is maintained for each complex of each rule, and this sum is augmented by the amount indicated by the tuple associated with the satisfied variable-value pair. The general formula for evaluating a complex is

AgAssistant: An Experimental Expert System Builder

$$\frac{\sum (\text{weights of satisfied selectors})}{\sum (\text{all weights in the complex})} \quad [1]$$

For example in complex1 of the general rule in figure 6., if variable1 had the value of value1 , and variable2 took the value value3, while variable3 the value value9 (not in the linear range value4 to value6) then complex one of this rule would have the value of:

$$\frac{60 + 40}{60 + 40 + 20} \quad [2]$$

or 83%. Selectors with internal disjunction (selector with a set of disjunctive values) are assumed satisfied if any of their values is the current value of the variable. Selectors with linear variables are satisfied if the variable takes on any value within that range inclusive of the end points. Selectors involving numeric ranges are satisfied if the answer given falls within the specified range. Similarly, selectors that are arithmetic expressions are satisfied if the expression evaluates to true. Finally, selectors with structured variables are true if the variable receives the indicated value or some child of that value.

Evaluation of rules with multiple complexes is performed by recursively taking the probalistic sum (referred to as 'psum' evaluation in Advise [Michalski and Baskin, 1983]) of the nth complex with the psum of the first n-1 complexes. The formula for this is:

$$\text{psum}(V(n), V(n-1), \dots, V(1)) = V(n) + \text{psum}(V(n-1), \dots, V(1)) - V(n) * \text{psum}(V(n-1), \dots, V(1)), \quad [3]$$

where $V(x)$ is the evaluated value of complex x

Other evaluation schemes such as taking the complex with the highest value are possible but are not included in this version of the system. The virtue of the psum scheme is that it fits well to the intuitive notion that if any complex is completely satisfied, the rule will also be, as well as the mathematical notion that the probability of two independent events occurring is the probalistic sum of the event probabilities.

A rule is considered to be satisfied if its confidence level goes above a threshold, currently set at 85%. Notice that if all selectors are assumed to have equal weights and the threshold is set at 100%, then the evaluation schemes described here collapse into formal deduction.

AgAssistant: An Experimental Expert System Builder

2.2.2 Evaluation of rules in a hierarchical rule base

Selectors in rules which are in turn actions of another rule receive the value of the rule times the weight for that selector. Thus in figure 3 (page 5), if one assumes that the sky is sunny in the third rule, and that the confidence that the weather is good is 50%. The confidence that the weather is good in the first rule is $50\% \times 40\%$, or 20%.

A hierarchical rule base contains a partial ordering between the rules. The rule evaluation module topologically sorts the rules according to this partial ordering before the advisory session begins, and uses the resulting order to evaluate all rules after each new value is entered. This ensures that all weights are propagated in the correct sequence. Notice, that if one assumes that all selectors of all rules receive equal weights, and if the threshold of rule firing is 100%, the inference scheme emulates a standard forward-chaining inference engine.

2.3. Control mechanism

AgAssistant has two control mechanisms, that is, schemes which determine the order in which questions are asked. The first method applies to rule bases which are flat, or non-hierarchical in nature, and is known as the utility scheme. The second mechanism, for hierarchical rule base, is a backtracking scheme.

2.3.1 Utility control scheme

The utility control scheme is pictured above in Figure 7. At the start of the advisory session questions are asked for variables in order of the utility of the variables. The utility measure, precalculated and found in the compiled system file, reflects the degree to which the variable will affect the confirmation level of all rules. Those variables appearing in the most places in the rules are given the highest utility. The advisory session continues until the confirmation level of a rule goes above a threshold, currently set at 15% for the WEEDER system. When this occurs, the system will focus on that rule by asking for the values of all variables relevant to the rule. This continues until the rule is rejected, or all variables for that rule are exhausted. At this point the system will focus on another rule which is above the threshold, or if none exists it will return to the utility measure. The entire process continues until all rules are rejected or confirmed. This may require that all variables be queried for, but usually occurs much sooner. In general, a rule is confirmed if it has a confidence level above the confirmation threshold, while it is rejected if it cannot possibly be confirmed, no matter what the values for the remaining rules are. The system makes use of the method of keeping a "negative" confidence for all rules; thus there is no need to dynamically determine whether a rule is rejected after each new answer. Rather, the rule is rejected if its calculated negative confidence exceeds $100 - (\text{threshold of confirmation})$. For example the WEEDER system has a confirmation threshold of 85%, therefore, a rule is rejected in the system if its negative confidence exceeds 15%.

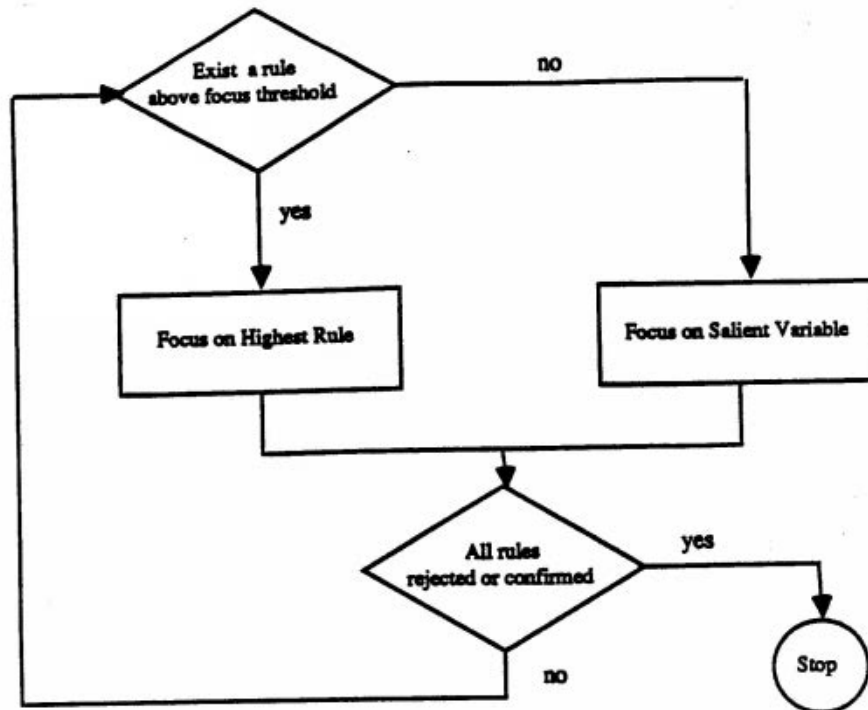


Figure 7. Utility control scheme

2.3.2 Backtracking control scheme

The backtracking control scheme is automatically invoked if the rule base is hierarchical. It is pictured in Figure 8. The system begins by asking question for variables with highest utility and continues in this fashion, until the user answers 'don't know' for a variable which also appears in the action of some rule in the system. For example, if the user answers 'don't know' for the variable *Weather* in the rule base of figure 3, then the system will attempt to infer the value of *Weather* by asking for the values of *Sky*, *Soilmoisture*, and *Windstrength*. The process continues recursively until the system is able to find the value for the original variable it was focusing on, in this case *Weather*, at which point it returns to the utility scheme. The system will continue to query the user until all rules are either rejected or confirmed.

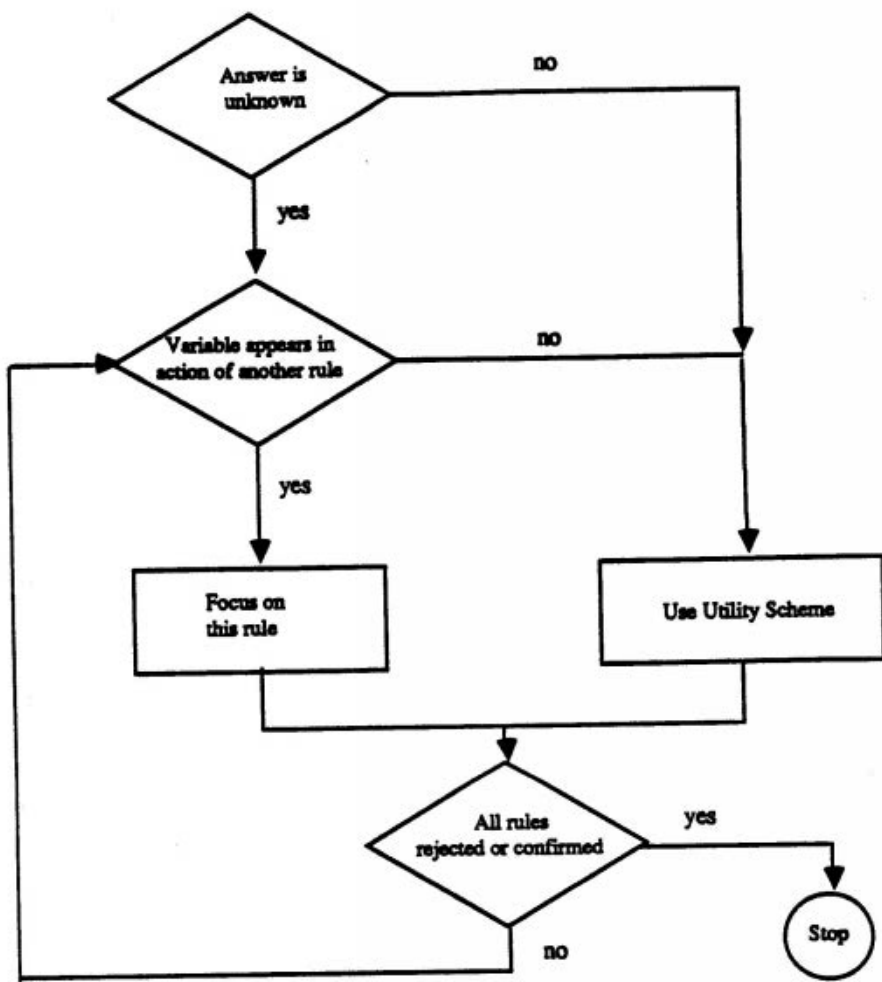


Figure 8. Backtracking Control Scheme

3. Knowledge Acquisition Facilities

Figure 9 illustrates the knowledge acquisition facilities of AgAssistant. As stated in Chapter 1, knowledge is represented in the system in the form of rules in modified VL1 syntax. These rules can be acquired in four possible ways. They can be learned from examples, improved with examples, optimized, or edited directly. It is also possible to use these methods in combination. For example, one common procedure is to edit rules directly, and then optimize them to remove superfluous information. Once acquired, rules are compiled to produce an advisory system. The various facets of knowledge acquisition are described below.

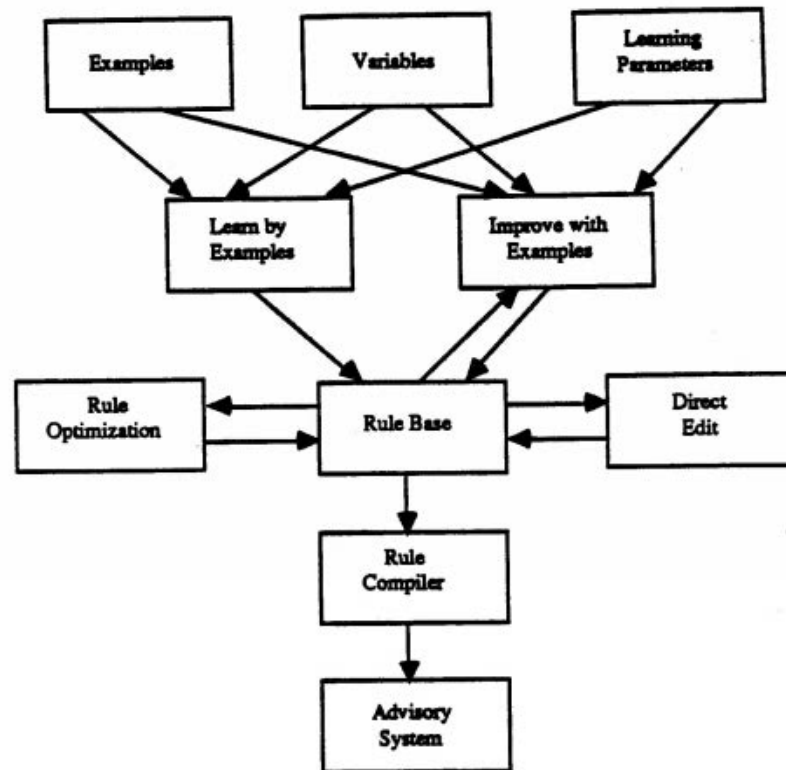


Figure 9. Knowledge Acquisition in AgAssistant

AgAssistant: An Experimental Expert System Builder

3.1 Direct Editing of Rules

The simplest way to enter knowledge into the system is to enter it directly. The necessary condition for this method is an expert capable of expressing knowledge in the form of rules. This notoriously difficult problem is often referred to as the *knowledge acquisition bottleneck* [Michalski and Chilausky, 1980b]. For example, one may be a perfectly adequate driver and yet have difficulty expressing this knowledge in rule format. Nevertheless, there are many domains in which direct entry of rules is appropriate. In AgAssistant, this is accomplished in two steps. First, the relevant variables are entered into the variable editor. The variable type, and its domain, are specified within the editor. Then, upon selection of the direct edit function within AgAssistant's menu system, a child process will be created. This process consists of the default editor (whose name is stored in the file 'ageditor.txt') with the file to edit being concatenation of the current system name and the extension '.rie'. The command processor of MSDOS will return the user to the AgAssistant system upon termination of the edit. The user will then typically attempt to compile the newly created rule base, and will repeat the edit-compile cycle if errors are indicated by the compiler.

It is important to note that the only way that AgAssistant can currently acquire a hierarchically structured rule base is by directly creating it. This is because the learning programs used in the system are not capable of constructing a knowledge base with intermediate layers of knowledge. They can only make correlations between the input events and their respective decision classes. One can partially work around this, if desired, by learning a set of subconcepts, and then learning higher-order concepts or directly entering such concepts. For example, one could first learn a set of rules describing diseases that afflict a given species. One could then do another experiment to learn the best treatment for the plant with one of the variables in this experiment being the disease, if any, of the plant. One could then concatenate the two resulting rule sets within the editor to produce a structured knowledge base. This method can be used to produce a rule group of arbitrary depth, although it is likely that any complex domain will consist of a mixture of expert and induced rules.

3.2 Rule Learning

At the heart of all the learning facilities within AgAssistant sits the NEWGEM program [Reinke, 1984], and at the heart of NEWGEM is the Aq quasi-optimal covering procedure. As Aq is described in detail in many previous papers, [see e.g. Michalski, 1973] we will not go into great detail about it here. Suffice it to say that Aq works by attempting to find a rule which covers all of the positive events and none of the negative events, positive events being those belonging to the decision class under consideration, and negative events being all others. It does this by selecting a seed event within the set of positive events and *extending it against* successive negative events until it covers none of the negative events. Extending a partial cover against a negative event simply means special-

izing it so that it no longer covers the negative event if indeed it did to begin with. This process is continued until a cover or disjunction of covers is produced for all of the original positive events.

3.2.1 Learning Rules from Examples

Learning from examples, is one of the most explored areas in Machine Learning [Dietterich and Michalski, 1983]. In this form of learning, a teacher provides characteristic examples and their respective decision classes to the learner. The task is then to create a set of rules which classify the given events. While simple in principal, this method of building descriptions of concepts is often powerful in practice. For example, in a now famous case it was shown that inductively derived rules for soybean disease diagnosis outperformed expert given rules [Michalski and Chilausky, 1980a]

AgAssistant combines variable definitions as found in the variable table, events as found in the data table, and parameters as set in the parameters table to form the file 'special.gem' which is then passed to the NEWGEM learning program. The parameters for the program determine various aspects of the rule creation process, including the breadth of the beam search used by the Aq algorithm, the *lexicographic functional* which is used in sorting candidate hypotheses, and the extent to which produced rules will be trimmed (see [Reinke, 1984] for more information on the meaning of the NEWGEM parameters, and the User's Guide in chapter 6 for information on setting these parameters). The NEWGEM module takes the input from 'special.gem' and sends its output to the file 'last.gem', which is then copied to the file '{system name}.rie' if the user decides to save the produced rules.

Each selector in a produced rule is associated with a weight. These weights are calculated by the following formula, and then normalized so that the weights of a given complex sum to 100.

$$\text{weight} = \frac{pe}{pe + ne} \quad [4]$$

where pe is the number of positive events covered, and ne is the number of negative events covered by the selector.

Thus, the weight produced represents the probability that the given decision class is indicated given that the selector is satisfied.

3.2.2 Improving Rules with Examples

AgAssistant is capable of improving its knowledge as new examples are presented to it. This method is known as incremental learning with perfect memory, and the algorithm for performing this task is presented in detail

AgAssistant: An Experimental Expert System Builder

in Reinke, 1984. Summarizing his description, we find the method to be a straightforward extension to the Aq algorithm. First, the existing cover for a class of events is specialized to take into account new negative examples. This new cover is then used as the original seed event for Aq.

As can be seen in figure 9, the input for rule improvement is constructed from the existing rule base, the table of examples which includes the newly entered examples, and the variable table. This file is then sent to the NEWGEM learning program which detects the presence of input hypotheses and therefore runs Aq incrementally. As Reinke mentions, one must be careful when using this learning mode, since it is likely that the complexity of the output rules will increase, although the amount of this increase depends on the nature of the new examples. Clearly, if the new examples are for the most part in existing decision classes, the rules will not change that drastically, assuming that the original induction was performed on a statistically large enough sample of events. If the new events fall into new classes, or if the original events came from a small subsection of the true problem space, then the rules will exhibit a proportional increase in complexity. The chief advantage of the incremental learning method presented here is the speed increase of the induction process.

3.2.3 Rule Optimization

In the context of AgAssistant, rule optimization refers to conversion of characteristic rules to discriminant rules. Michalski defines characteristic rules as those that specify common properties of the members of a class, and discriminant rules as those which have only enough information to distinguish one class from another or another set of classes. [Stepp, 1983] Here characteristic rules will refer to any that are not discriminant. For example, expert-created rules typically fall somewhere in-between the discriminant and characteristic categorizations. Indeed, one common use for this facility is to compress rules provided by an expert to their discriminant versions.

The method for rule optimization takes advantage of facilities already provided by the NEWGEM learning program [Reinke, 1984]. Rules are converted to the proper internal format for NEWGEM and submitted as input hypotheses with no corresponding input examples. If the rule type parameter is set to produce discriminant rules, these rules will be generalized to discriminant form.

Figure 10 shows the results of converting three characteristic rules to discriminant form. Notice that the optimized rules contain only the information necessary to distinguish between the three actions, in this case, the values of the variables shape and texture. This method of rule optimization only makes sense in context of an expert system if one is confident that the values of the discriminatory variables will be known by the user of the expert system. If this is not the case, the system will perform better if the rules are left in their characteristic form.

Before optimization

Action is one if:

1. Color is red,
2. Size is small,
3. Shape is round,
4. Texture is smooth.

Action is two if:

1. Color is red,
2. Size is small,
3. Shape is square,
4. Texture is rough.

Action is three if:

1. Color is red,
2. Shape is small,
3. Shape is square,
4. Texture is smooth.

After optimization

Action is one if:

1. Shape is round,
2. Texture is smooth.

Action is two if:

1. Shape is square,
2. Texture is rough.

Action is three if:

1. Shape is square,
2. Texture is smooth.

Figure 10. Rules before and after optimization

3.3 Rule Compilation

Once acquired, rules must be compiled if they are to be used in an advisory capacity. The compiler has two chores. One is to check the syntax of the rules and provide the user with the appropriate error messages if the rules do not parse. (A complete listing of error messages is provide in section 6.5). If the rules parse successfully, the compiler will then create an file suitable for asking questions and giving advice. The exact nature of this file is detailed in section 2.1 and will not be repeated here.

Figure 11 below contains the complete grammar for rules in the system. Summarizing this figure, we see that a rule consists of a condition part and an action part. A condition consists of the disjunction of a set of complexes, which in turn consist of the conjunction of selectors. A selector consists of a variable, a relation, and either a value, a disjunction of values, or a range of values, plus an optional weight. A selector may also consist of an arithmetic expression.

The rules are parsed in a straightforward way by an ATN-like parser. That is, the parser is structured as a finite-state machine, with each of the elements of the machine optionally being another sub-machine. The only real complication comes in the handling of arithmetic expressions. All arithmetic expressions are converted to reverse Polish notation to ease the checking of the syntax. A similar conversion is performed at the time of running the

AgAssistant: An Experimental Expert System Builder

advisory system, to allow the expression to be evaluated by recursive descent through the previously developed tree. Currently, the only permissible operators in arithmetic expressions are plus, minus, multiplication and division. An arithmetic expression may also consist of a single real number. The parser recognizes arithmetic expressions within a selector only if the relation is one of "<", "<=", "=", ">", or ">=".

<rule>	:=	<action> <condition>
<action>	:=	<variable> <relation> <value> "if:"
<condition>	:=	<complex> "OR" <condition>
	:	<complex>
<complex>	:=	<selector> <complex>
	:	<selector>
<selector>	:=	<variable> <relation> <value-list> <terminator>
	:	<a-expression> <a-relation> <a-expression> <terminator>
<variable>	:=	string of letters
<relation>	:=	string of letters
<value-list>	:=	<value> "or" <value-list>
	:	<value> "to" <value>
	:	string of letters
<terminator>	:=	<termchar>
	:	<termchar> <weight>
<termchar>	:=	" "
	:	"."
	:	"."
<weight>	:=	natural number
<a-expression>	:=	"(" <a-expression> <a-operator> <a-expression> ")"
	:	<variable>
	:	<r-number>
<a-operator>	:=	"+", "-", "*", "/"
<r-number>	:=	real number
<a-relation>	:=	"<.", "<=", "=", ">.", ">="

Figure 11. Grammar for Rules

4. An Exemplary System: WEEDER

In the design of an effective weed-control program for turf, it is first necessary to correctly identify the species of weed(s) present and to determine the extent of their population. Morse (1971) outlines five basic identification methods for determining unknown plant species: i) *Expert determination*, which is generally regarded as the most reliable of all identification techniques. This method merely transfers the responsibility of identification to an appropriate expert. This service can be slow and costly, and is often limited by the availability of an expert. ii) *Immediate recognition*, approaches expert determination and accuracy. This is the ability of an individual to recognize an unknown weed by past examples of identification. For some taxonomic groups and immature plants, however, this method of identification is very difficult and in all cases requires extensive past experience. iii) A *comparison of an unknown specimen with identified species* or illustrations. It offers a rapid, simple diagnosis and is often useful for many weeds commonly found in native populations. iv) An *identification key* which is based on the development of appropriate descriptive phrases of morphological or biochemical characteristics. Identification keys generally take the form of groupings of similar characters from which the user must select the character which best matches that present on the unknown sample. The selection of this character then leads to the next set of identifying characteristics. This process is followed until enough characteristics have been identified to suggest the identification of the specimen. v) The last identification technique is a *diagnostic table or polyclave*. Diagnostic tables are a matrix of rows of species and columns of identifying characteristics. Users of a diagnostic tables can identify the listed characteristics in any order they wishes.

Morse (1971) lists two major faults of identification keys: i) They require a user to utilize certain characteristics whether or not they are convenient or can be identified; ii) They implicitly rely on rigid descriptions of specimens. Occasional variation in a population can cause gross misidentification.

The use of expert systems techniques offers a new, unique method for assisting with species identification. The relative merits of an expert or advisory systems is the ability to select answers or queries about characters that are available on the unknown specimen. They can operate on various levels of uncertainty providing a more efficient mechanism for identification, particularly for immature plants which are even difficult for experts to identify.

4.1 Development of WEEDER

In order to prepare the knowledge for use with AgAssistant, a matrix was developed including each potential grass weed. Eleven identifying characteristics, both vegetative and floral were determined for each weed. The information for this table was obtained from many sources: textbooks, weed identification manuals, botanical manuals, and the authors experience. The variable name, type, and set of values is shown in Figure 12.

AgAssistant: An Experimental Expert System Builder

VARIABLE	Vernation	Auricle	Ligule	Sheath	Collar	Blade_width
TYPE	nominal	nominal	nominal	nominal	nominal	linear
1	folded	absent	ciliate	compressed	narrow	fine
2	rolled	short	round	round	divided	medium
3		claw_like	truncate	closed	broad	coarse
4			acute			
5			toothed			
6			acuminate			
7			none			
8						

VARIABLE	Habit	Glumes	Disart	Awms	Florets	Flower	Nerves
TYPE	nominal	nominal	nominal	nominal	integer	nominal	integer
1	bunch	shorter	above	absent	1	panicle	1
2	rhizome	longer	below	present	3	raceme	3
3	rhiz_stolon			bifid	5	spike	5
4	stolon						
5							
6							
7							
8							

Figure 12. Variable names, types, and values for WEEDER

The characteristics selected were those thought to be most easily recognized in the field without supportive equipment. [Shurtleff, et al., 1987] Figure 13. presents a typical rule. This rule consists of 10 selectors. Each selector is associated with a weight or confidence level (cl) that indicates the relevant importance of the condition in making the decision. Notice that these weights need not add up to 100; they are normalized by the system. The weights give a rough estimate of the importance of each of the conditions in discriminating between the rules. These weights were then refined by the domain expert (T. Fermanian)

Rules for WEEDER were developed utilizing the NEWGEM module of AgAssistant. Separate rule sets were formed, first by inducing a set of characteristic rules, and then by inducing a set of discriminate rules. The most appropriate rules from both sets were then modified, utilizing expert experience and written to a single rule set used in the initial evaluation.

Grass weed identification in turf is generally only available through the use of vegetative characteristics. This is due to the frequent mowing of the turf which often removes any floral portions of the plant. WEEDER allows the user to select either vegetative or a combination of floral and vegetative characteristics at the beginning of

AgAssistant: An Experimental Expert System Builder

each session. This is done through a "does-not-apply" question which is always asked first in the consultation.

Weed is Stinkgrass if:	cl
1. Florets are 10 to 12,	85
2. Flower is panicle,	70
3. Collar is narrow,	60
4. Blade_width is medium,	55
5. Habit is bunch,	50
6. Sheath is compressed,	50
7. Vernation is rolled,	35
8. Glumes are shorter,	30
9. Florets is 1,	30
10. Disart is below.	25

Figure 13. A rule for identifying Stinkgrass

"Does not apply" questions were established in order to provide a meaningful subset of variables for WEEDER to act on. The question "Are seedheads or flowers present?" to which the user responds "yes," "no," or "don't know" begins each session. If a "don't know" answer is given, then all identifying characteristics are asked. If "yes" is answered, then eleven of the possible characteristics are presented to the user. If "no" is answered, which is the usual situation for turf, then seven characteristics are presented—only those pertaining to vegetative portions of the plant. Paraphrased versions of these do not apply conditions are shown in figure 14.

If seedheads are present then

Auricle, and Blade_width do not apply.

If seedheads are not present then

Florets, Flower, Awns, Disart, and Glumes do not apply.

Figure 14. Does not apply conditions in WEEDER

AgAssistant: An Experimental Expert System Builder

4.2 Validation of WEEDER

In order to measure the relative efficiency of WEEDER in identifying unknown grasses, a study was conducted in which individuals were asked to identify four unknown grasses. Four grasses were selected randomly from a set of fifteen grass species commonly found in central Illinois. The four species selected were: creeping bentgrass (*Agrostis palustris* L.), perennial ryegrass (*Lolium perenne* L.), zoysiagrass (*Zoysia japonica* L.), and large crabgrass (*Digitaria sanguinalis* (L.) Scop.). Forty-one volunteers were assigned to one of two groups, if they had had any previous experience in plant diagnosis or any formal training in plant science, they were separated from those volunteers who had no biological or plant science training or experience. Each individual randomly selected two of the four unknown weeds for identification using WEEDER, the other two weeds were identified using a diagnostic key, a commonly used tool (Shurtleff et al., 1987).

Along with the four unknown grass samples, each participant was supplied with a low-power dissecting microscope, appropriate probes and dissecting equipment, and a book with representative diagrams of all the potential configurations of morphological characters. Each individual was allowed up to 30 minutes per weed for identification. Fifteen minutes was reserved for a demonstration of each character and an explanation of how it could be identified. For the plants identified through the diagnostic key, each participant only supplied their first and possibly a second choice, as suggested by the key. Grasses identified with WEEDER, however, offered participants the ability to indicate the configuration chosen for each plant character. A frequency analysis of the correctly identified grasses was then conducted to determine their fit to the χ^2 distribution.

Based on the consistency of participant identified configurations of several characters for each plant, rules representing the four unknown species were *adjusted* by changing the CL values and in some cases, adding local disjunctions of additional configurations. The second set of correctly identified grasses, occurring after rule adjustment, were then evaluated for their fit to the χ^2 distribution as previously described. The results of both evaluations is shown in table 1.

WEEDER has the ability to rank all the grasses in its knowledge base from most likely to least likely representing the unknown grass. This could potentially provide 39 answers or identifications for each unknown sample. Table 1 presents the percentage of correctly identified grasses listed as either the first or second choice using either identification tool (WEEDER or the key.) Regardless of the tool, the level of performance was quite low. As expected, a tool commonly used by the participants with a plant science background, the identification key, showed a modest success rate (22%.) The 15% success rate of the plant science group in correctly identifying the grasses using WEEDER was initially disappointing. While the 7.1% success rate for the non-plant science group using WEEDER was considerably less, it appeared to be independent with a probability of .8.

AgAssistant: An Experimental Expert System Builder

Table 1. Percentage of correctly identified grasses in either the first or second choice using WEEDER or a key.

Identification Tools	User groups					
	Plant science Background			No previous Training		
	Both groups		Both groups		Both groups	
	Rule readjustment		Rule readjustment		Rule readjustment	
	Before	After	Before	After	Before	After
	%					
WEEDER						
Bentgrass	12.5	62.5	9.1	54.5	10.5	57.9
Per. ryegrass	12.5	37.5	9.1	45.5	10.5	42.1
Zoysiagrass	16.7	75.0	0.0	80.0	9.1	77.3
Large crabgrass	16.7	41.7	10.0	0.0	13.6	22.7
All grasses	15.0	55.0	7.1	45.2	11.0	50.0
χ^2	.13	3.97	1.02	13.52	1.29	.78
P	.99	.26	.80	.004	.26	.38
Id key						
Bentgrass	16.7	— ^t	10.0	—	13.6	—
Per. ryegrass	41.7	—	40.0	—	40.9	—
Zoysiagrass	25.0	—	10.0	—	15.8	—
Large crabgrass	0.0	—	10.0	—	5.3	—
All grasses	22.5	—	17.5	—	19.5	—
χ^2	7.11	—	2.65	—	1.50	—
P	.07	—	.45	—	.22	—
Both tools						
	18.8	40.0	11.9	29.8		
χ^2	1.25	7.50	1.12	9.62		
P	.26	.01	.29	.002		

^tnot readjusted

Rules for the four grasses were modified to raise the level of confidence of the more readily identified characters and to reflect the strongly identified configurations which had not been included in the original rules. The results of an analysis of these changes showed a *large* gain in the percentage of correctly identified grasses as shown in Table 1. On the average, the percentage of correctly identified grasses rose to 50%, as compared to the 19% for grasses identified with the key. A χ^2 evaluation showed no significant dependence on previous training when using WEEDER. A summary of the independence of correctly identified grasses is shown in Table 2. While no significant indication of dependence was shown before rule adjustment, a very significant dependence (.00) indicates the

AgAssistant: An Experimental Expert System Builder

relative advantage of WEEDER over the identification key. The key appeared to be equally useful to groups with either training, showing no significant dependence before or after rule adjustment.

Table 2. Summary of the independence of correctly identifying four grasses as the first or second choice using WEEDER

Rule readjustment	Identification tool	User group
Before		
χ^2	2.31	2.74
P	.13	.10
After		
χ^2	16.8	1.89
P	.00	.17

One of the most prominent findings of this investigation was the relatively poor performance in identification of unknown grasses by individuals regardless of their background. When using the key, performance was generally better from the group with the plant science background. This difference in performance, however, was not found when the same group used WEEDER, which benefited either group equally. It is important to note that a significant gain in the ability to correctly identify weeds was found with WEEDER, particularly after rules were adjusted. This study brings out one important aspect to the use of expert advisory systems. While the use of knowledge is central to all advisory systems, the skills associated with recognizing queried characters is paramount in weed or species identification. These recognition skills were lacking in the test population, as indicated by the low ratios of success.

While WEEDER provided an initial test of AgAssistants inferencing capabilities, other portions of the program remain untested (learning module, rule optimization, etc.). Additional systems are currently being developed to test these functions.

5. Conclusions

An expert system builder that is capable of learning and improving its knowledge has been presented. Thus it has been demonstrated that sophisticated knowledge acquisition facilities are suitable for creating expert systems in the microcomputer environment. This should be of great use in disseminating this technology to the typical agricultural user who does not have access to large computers

A number of improvements and extensions to AgAssistant are possible, however.

- 1) New and more powerful learning systems could be incorporated into the system. One such method is learning by analogy, in which the program acquires knowledge by comparison with similar cases it has seen in the past.
- 2) The ideal system should be able to adjust its knowledge during the advisory session. That is, if it is told that it made an incorrect decision, it should be able to update its knowledge in light of this information.
- 3) The current system uses a very simplistic method for combining evidence. The creator of the system should be able to state the importance of groups of conditions in addition to weighting individual conditions.
- 4) Automated methods for generated explanation and other text during the advisory session need to be developed.
- 5) The learning module should be able to incorporate background knowledge. In addition, it should be able to suggest a hierarchical structure in addition whereby input events are connected to decision classes through intermediate nodes.
- 6) The programs CLUSTER [Stepp, 1983], for clustering examples into categories, and ATEST [Michalski, 1985], for testing the consistency and completeness of rules could be incorporated into the system.
- 7) Finally, the system could be integrated with a video system. This would enable the system to display plants and other items during the question answering phase of the advisory session

This list provides suggestions for further research in this area.

6. AgAssistant User's Guide

Figure 15 below gives an overview of the AgAssistant from the point of view of the system user. From the main menu of the system, one may either get advice from an expert system, or develop an expert system. This chapter explains in detail these options.

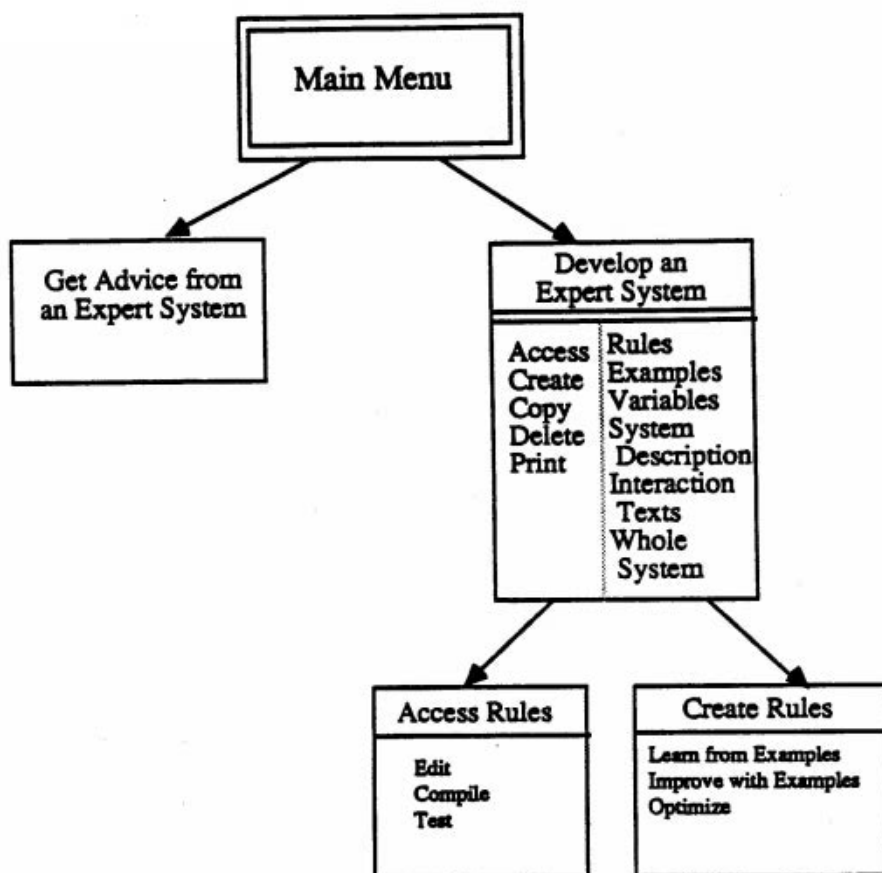


Figure 15. An Overview of the AgAssistant Menu Structure

6.1 The Main Menu

Entering 'agr' from the operating system will invoke the AgAssistant system, and bring up the main menu [Figure 16] for the system. Entering 'agr demo' will do the same after greeting the user with a short animated display.

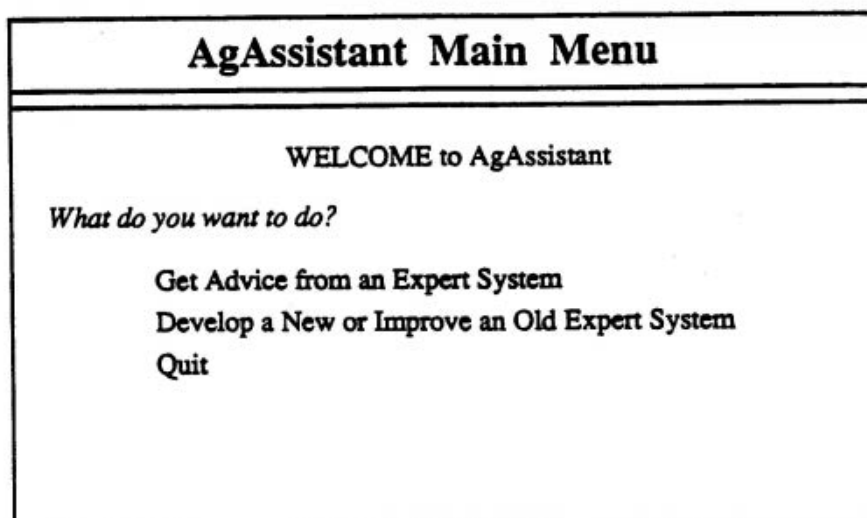


Figure 16. The Agassistant Main Menu

The main menu displays the three major options one can choose from the start of the program. As with all menus in the AgAssistant system, options are selected by moving the 'box' which highlights a selection to the one that you desire, and then typing RETURN (↵) to activate it. Movement of the box corresponds in a natural way to the direction keys located in the numeric keyboard to the right of the PC keyboard. The user may also type SPACE to move the box forward, or BACKSPACE to move the box backward. The options for this menu are:

1) *Get Advice from an Expert System*

The user chooses this option to get advice from an existing expert system. One cannot modify the system in any way with this option. This selection will bring you to the menu described in section 6.1.1.

2) *Develop a New or Improve an Old Expert System*

One selects this option if one wishes to modify an existing system, or to build a new intelligent system

AgAssistant: An Experimental Expert System Builder

from scratch. This option provides access to all of the knowledge editing functions of AgAssistant, as well as the learning facilities. This choice brings you to the menu described in section 6.3.

3) *Quit (Leave the program)*

6.1.1 Getting Advice from an Expert System

After choosing the option, 'Getting Advice from an Expert System', from the Main Menu, the screen pictured in figure 17 is displayed. Notice that the top line of the screen tells you what you are doing now (on the right), and which menu you came from (to the left). This convention is followed throughout AgAssistant.

Getting Advice		Came from: Main Menu	
Expert System		Description of Expert System	
Domain: Weed Identification Weeder Weed1 Weed2		Expert defined rules for weed identification Characteristic rules for weed identification Discriminant rules for weed identification	
Domain: Turfgrass Management Golf Field		Selecting and maintaining turf for golf courses Selecting and maintaining turf for playing fields	
Options:		Page 1 of 1	
PgUP - Previous page of Systems		RETURN - Select system	
PgDN - Next page of systems		ESC - More Options	

Figure 17. Getting advice from an Expert System

The expert systems and their domains are displayed on the left of the screen. The right hand side contains a brief description of the expert systems. Options for this menu are:

AgAssistant: An Experimental Expert System Builder

1) *PgUp - Previous page of systems*

If there are more than one pages of expert systems, this option will bring one to the previous page, or sound the bell if one is already on the first page. A page count appears on the right hand side of the screen adjacent to the word "Options:".

2) *PgDn - Next page of systems*

If there are more than one pages of expert systems, this option will bring one to the previous page, or sound the bell if one is on the last page.

3) *RETURN - Start an advisory session*

Move to the desired system with the direction keys, or the paging keys. When the system you desire is highlighted, type RETURN. This will bring start the advisory session (described in section 6.1.2.).

4) *? - Learn more about a system*

After a given expert system is highlighted, typing '?' will bring up one or more pages of information about that particular system.

5) *ESC - More options*

Typing ESC at any time will bring up further options that allow one to either return to the main menu, or to quit the program. This convention is followed throughout AgAssistant.

6.1.2 Getting Advice from a particular system

After selecting the desired system, a series of questions will be put to the user of the program in order to determine what course of action to follow. Questions are in two basic formats, those requiring the user to select an answer, and those in which the user is expected to type in a numeric answer. The appropriate methods of response to these questions are explained below.

Figure 18 contains a sample screen in which the user is being asked for the appearance of the ligule (morphological character of a grass). The choices in this case are ciliate, round, truncate, acute, toothed, acuminate, absent, and don't know. An answer is selected simply by moving the box to the required entry with the direction keys and pressing RETURN. Additional options are explained below in section 6.1.3 All questions permit the user to answer that he does not know that particular answer. If the system can, it tries to determine the value through alternative means, otherwise it simply goes onto the next question.

AgAssistant: An Experimental Expert System Builder

Getting advice from Weed	Came from: Selecting a system
Plan	Answers so far
Focusing on hypothesis: Weed is Yellow Fxtail	4. Blade width is fine 5. Awns are absent 6. Flowers is panicle 7. Nerves are 0 Rules with highest confirmation 1. Weed is Yellow Fxtail 73% 2. Weed is Ani Bluegrass 30% 3. Weed is Alkaligrass 28% 4. Weed is Rescuegrass 28%
How does the ligule appear?	
ciliate truncate toothed absent	round acute acuminate don't know
Options:	
RETURN -- Select Answer ? -- Help with an answer	ESC -- More Options

Figure 18. Responding to a questions requiring an user to select an answer

In some cases, a numeric answer is required. The user will be requested to type in a real number within the specified range. For example, in figure 19, the following are valid answers: 3.9, 8.14521, 0, 12.0. the following are not in the range specified, and are invalid entries: -100, 12.1, xyz, and 8b. The user may also type RETURN only, with no characters appearing in the field, if he does not know the answer to the question.

6.1.3 Options while answering a question

1) RETURN - enter answer

After highlighting the appropriate answer by using the space bar, or entering a numeric value, typing RETURN will tell the system to accept that value, update its knowledge, and move on the the next question.

Getting advice from Weed		Came from: Selecting a system	
Plan		Answers so far	
Focusing on variable: pH		Rules with highest confirmation	
What is the pH of the soil? _____ < Range: 0-14 > (If you don't know the answer, type RETURN only.)			
Options:			
RETURN -- Select Answer		ESC -- More Options	
? -- Help with an answer			

Figure 19. Responding to a questions requiring a numeric answer

2) ? - Help with an answer

This option will display more information about a highlighted answer, if such information was entered during system development.

3) ESC - More options

Typing ESC will bring up a number of options that allow one to move to other menus. These are:

a) View Confidence Levels

If chosen, this option permits one to view the confidence level of every rule in the expert system, as illustrated in Figure 20. Notice that across from each rule, a confidence level is displayed both numerically and graphically. Rules are divided into three classes. Rules that are confirmed are those rules above the

AgAssistant: An Experimental Expert System Builder












Viewing Rule Confidence Levels		Came from: answering questions	
Rule Action	0	100 %	
Confirmed rules:			
Weed is Yellow Fxtail			73
Unconfirmed rules:			
Weed is Stinkgrass			40
Weed is Ani Bluegrass			30
Weed is Alkaligrass			28
Weed is Rescuegrass			28
Weed is Fall Panicum			27
Weed is Ken Bluegrass			24
Weed is Smutgrass			23
Weed is Redtop			21
Rejected rules:			
Weed is Lrge Crbgrass			28
Weed is Torpedograss			17
Options:		Page 1 of 4	
PgUP -- Previous Page of Rules		Return -- Return to answering questions	
PgDN -- Next Page of Rules			

Figure 20. Screen for viewing confidence levels

confirmation threshold, currently set at 85%. Rules that are unconfirmed are those rules not above this threshold, but with a chance of being such depending on the answers to further questions. Rules that are rejected are those that have no chance of being confirmed. Note that it is possible for a rejected rule to have a higher confidence level than one which is unconfirmed. Options from this menu include paging through the rules, or returning to answering questions.

b) View a Rule

This option permits one to view any rule in the system. One will be asked to type in a rule name, or to type "?" to select from the rules listed. The entire body of the rule will then be displayed. This is useful for

informing the user of the conditions which are associated with a rule and why the confidence is at a given level.

c) Start a new advisory session

Begin a new session and lose all answers entered in this one.

d) End this advisory session

End the session prematurely. Normally, a session will continue until all rules have been rejected or confirmed. This option allows one to terminate it before this time. One will be brought to the screen described in section 6.1.1.

e) Resume this advisory session

If ESC is typed by accident, or you merely wished to view the available options, you may select this option to restart the session where you left off.

f) Quit

Quit the program.

6.2 The Windows

Accompanying each question screen are three windows to aid the user in his understanding of what the system is doing. The Plan Window describes to the user why a particular question is being asked at this time. The Answers Window shows the user his answers to the last four questions. The Confirmation Window indicates the rules with the four highest confirmation levels.

6.2.1 The Plan Window

The Plan Window Consists of three sorts of messages to inform the user of the intentions of the expert advisor. Figure 21 shows examples of these messages.

In A) we learn that the system is focusing on a specific variable. This will typically occur at the beginning of an advisory session. When enough evidence accumulates to point to a specific rule, messages such as those found in B) will appear. In this case, the system first tried to confirm the hypothesis "Weed is Torpedograss" by asking a series of questions relevant to this rule. It was rejected, however, and the system is now concentrating a new rule

AgAssistant: An Experimental Expert System Builder

Plan	
A.	Focusing on the variable pH.
B.	The hypothesis 'Weed is Tropedgrass' is rejected. Now focusing on the hypothesis 'Weed is Stinkgrass'.
C.	Focusing on the variable temperature, In order to determine the value of weather, In order to determine the value of planting conditions.

Figure 21. Typical entries in the plan window

- A) The system is focusing on a variable.
- B) The system is focusing on a rule.
- C) The system is attempting to find the value of a variable indirectly

"Weed is Stinkgrass". Case C) applies only to hierarchically structured rule bases. In this situation, the user answered "don't know" to the question "What are planting conditions like". The system then backtracked and tried to infer the answer to this question by asking for a description of the weather. The user answered "don't know" once again, and the system is now focusing on the temperature in order to ascertain weather conditions.

6.2.2 The Answers Window

The Answers Window simply presents the user with the answer to his last four questions. for example, in figure 18 above, we see the previously entered values of blade width, awns, flowers, and nerves.

6.2.3 The Confirmation Window

The confirmation window lists the rules with the highest four confirmation levels. Note that this does not mean that these rules are necessarily confirmed absolutely. Rules cannot appear here, however, once rejected. Figure 18 above includes a typical Confirmation Window found in the upper half of the screen. Note that the rule actions are followed by their respective confirmation levels.

6.2.4 The end of an advisory session

When all hypotheses have been rejected or confirmed, or if the user artificially terminates the advisory session, a new menu will appear. A typical screen appears in Figure 22. The screen offers advice on the basis of the user's responses. This advice can take one of two forms. It can consist of a listing of one or more confirmed hypotheses, and their respective confirmation levels. Or, as in the case below, it may be a page of text associated with the rule with the largest confirmation level.

Advise Session Complete		Came from: Answering questions	
On the basis of your responses, I offer the following advice:			
The weed that fits closest to your description is Yellow Foxtail.			

Options:			
View Confidence Levels	Change an Answer	Main Menu	
View a Rule	Start Another Advise Session	Quit	

Figure 22. the final screen of the advisory session

Options for this menu are:

1) *View of confidence levels*

This will bring one to the screen in figure 19. One can then view the final confirmation levels of all rules.

2) *View a Rule*

This option is identical to the one during the advisory session, lets one see the full body of a rule.

AgAssistant: An Experimental Expert System Builder

3) *Change an answer*

Selecting this option allows one to change any answer given during the session, in order to see the corresponding change in the confirmation levels of the rules.

4) *Start another advise session*

Go to the menu in figure 17 above.

5) *Main Menu*

Go to the menu in figure 16 above.

6) *Quit*

Quit the program.

6.3 Develop a New or Improve an Old Expert System

After choosing the option 'Develop a New or Improve an Old Expert System', the screen in figure 23 appears. This menu allows one to access all of the knowledge acquisition facilities of AgAssistant. One uses this menu by selecting an item from each of the three columns on the screen. The header of each column is highlighted upon entry. The user moves from column to column with the right and left direction keys, and within a column with the up and down direction keys.

Typically, one will first select an expert system from the right most column. this may be an existing system, or the special entry *<New System>* for creating a new system. Next one moves right, and selects an operation one wishes to perform. Finally, one chooses an item from the right most column. A line below the columns indicates which particular combination of three items is currently being chosen. One activates the given choice by pressing RETURN. One may also type ESC at anytime to get options for changing the menu. An explanation of the possible choice one can make from this menu are detailed below.

One creates a new entry in the first column by accessing the system description of the *<New System>* (this is the only operation one can perform). When created, the newly created system name will appear in the list of systems, and one can then begin creating its components. Each expert system consists of a set of items in the rightmost column. The *Interaction Texts* entry is not yet implemented; it will contain information relevant to the advisory session. Likewise, the *Inference Parameters* column is not implemented; eventually, this will allow the user to select the control strategy and other control related information for the advisory session. The *Whole System* can only

Developing New or Improving Old Expert System		Came from: Main Menu
Select Expert System	Select Operation	Select Item
<New System>	Access	Rules
Domain: Turfgrass Management	Create	Examples
Golf	Copy	Variables
Lawn	Delete	System Description
Domain: Weed Identification	Print	Interaction Texts
Weeder		Inference Parameters
Weeder1		Whole System
Domain: Object Recognition		
Trains		
<<Selection: Accessing System Description of <New System> >>		
Options:		
Return - Get Selection		
ESC - Other Options		

Figure 23. Developing an Expert System

be copied or deleted. Throughout this section, a small sample expert system called "Trains" will be used for illustration.

6.3.1 Accessing Rules

Figure 24 displays the screen for accessing rules in a system. One has two choices, besides returning to the Develop Menu and quitting, from this menu.

1) Edit Rules Directly

This option takes one to the editor named in the file "Ageditor.nme" with the appropriate rules file passed to the editor (currently not implemented). This file is a concatenation of the expert system name and the extension ".rle". One can modify rules in any fashion while in the editor. The modified rules must be compiled before they can be used in an advisory system.

AgAssistant: An Experimental Expert System Builder

Accessing Rules in Weed	Came from: Developing an Expert System
Choose an operation: Edit Rules Directly Compile Rules	
Options:	
Developing Expert System Menu Quit	

Figure 24. Accessing Rules

2) Compile Rules

Invoking this option calls the system compiler. Rules are parsed for syntactic correctness and converted into a compressed form for advising. Figure 25. shows a screen which results from the successful compilation of the three rules in Train. Various errors may be found in the rules during compilation; a full list of them appears in section 6.5. In this case, a non-fatal warning was issued because the variable *Cargo* was not declared in the variable table. The system makes the default assumption that variables are nominal if they are not declared otherwise. The exact syntax of the rules is presented in section 3.3; chapter 7 shows the rules for the exemplary system WEEDER. Options for moving to other menus are available upon completion of the compilation.

6.3.2 Creating Rules

Creating rules invokes the learning module of AgAssistant. As shown in Figure 26, this menu allows one to create rules in three ways: by learning them from examples, by improving them with examples, and by optimization. One also has the option of editing the parameters that affect the learning process. These options, in more detail are:

Compiling Rules in Trains	Came from: Accessing Rules
<p>PARSING RULE: Direction is Northbound *****Warning***** Cargo not declared Assuming nominal type Northbound successfully parsed.</p> <p>PARSING RULE: Direction is Southbound Southbound successful parsed.</p> <p>PARSING RULE: Direction is Eastbound Eastbound successfully parsed.</p>	
Options:	
Rule Access Menu Developing ES Menu	Main Menu Quit

Figure 25. Rule Compilation

Creating Rules in Trains	Came from: Developing an Expert System
<p>Choose an operation:</p> <ul style="list-style-type: none">Learn Rules from ExamplesImprove Rules with ExamplesOptimize Rules According to Parameter SettingsEdit Learning Parameters	
Options:	
Developing Expert System Menu Quit	

Figure 26. Creating rules

AgAssistant: An Experimental Expert System Builder

1) Learn rules from examples

One selects this option only if one has examples created for the system as described in section 6.3.3. The screen in figure 27. will be displayed. One has four items to enter before starting the learning program. They are preset and need not be altered if one desires simply to learn from the current examples in the given system (trains, in this case), and output the rules to the system of the same name. One can optionally alter the name of the system containing the examples, the name of the concept to learn (default is all concepts), and the system to output the rules to. The latter option is especially useful since the new rules will overwrite the old system if they are saved. One starts the learning procedure by typing ESC and then choosing the option to *Start Learning*. The parameters used will be those set in the parameter table (the fourth option from the menu in figure 26.).

Learning Rules from Examples in Trains	Came from: Creating Rules
Example ES: trains (Enter ES containing examples or type '7' to select from event files)	
Rule ES: none (Cannot improve rules here)	
Concepts to learn: all (Enter single concept to learn or type '7' to select from concepts)	
Output to ES: trains (Enter ES to send rule to)	
Options:	
Start Learning	Return to Developing ES
Return to Creating Rules	Quit

Figure 27. Learning Rules from Examples

The learning module will first read in the learning events, and then display a screen similar to that shown in figure 28. In this case, the system has determined a rule for the Northbound class and is in the process of creating the Eastbound rule. The Northbound rule has four selectors, each associated with three figures in the columns to the right. The first column indicates the confidence level of that selector and is a function of the other second column, indicating the number of positive events covered by that selector, and the third column, indicating the number of negative events covered by that selector. The *Total* column shows the number of positive events covered by the complex, in this case 2.

Learning Train Rules				
Creating Eastbound Rule	Conf. Level	Events Covered		
		Pos.	Neg.	Total
Direction is Northbound if:				
1. Color is blue or green,	40%	2	1	
2. Car shape is box,	27%	2	1	
3. Size is small to medium,	16%	2	3	
4. Fuel is oil or gas.	16%	2	3	2
Options:				
<<Type ESC to abort learning program>>				

Figure 28. Screen displayed while learning rules

One can abort the learning procedure at any time by typing ESC. Upon aborting, or upon the completion of the learning procedure, the following options are given:

a) *View a Rule*

One can view any of the newly created rules.

AgAssistant: An Experimental Expert System Builder

b) Save and Return to Developing ES Menu

Save the new rules and return to the menu in figure 23.

c) Save and Compile

Save the new rules and compile them into an expert system.

d) Learning Rules Menu

Return to the menu in figure 26 without saving the rules. This option is useful if wants to repeat learning rules after adjusting the parameters.

e) Quit

Quit the program.

2) Improve Rules with Examples

One often wishes to take advantage of new examples to improve an existing rule base. The rule base may have been generated by any of the methods in the program. This option, however, will typically be selected when you want to add a few examples to the set of examples that were used to initially generate the rule base. The learning performed in such a manner will be much quicker than if one were to learn from the total set of examples from scratch.

The menu of figure 29 appears when this option is selected. It is identical to that of figure 27, except that one is also given the option of entering the name of a rule base to improve (the default is the same as the working system). After starting learning, the same screen as in figure 26 will appear. Options after learning is complete are identical to those in learning from examples.

3) Optimize Rules according to Parameter Settings

One often wishes to take a set of rules that characterize a set of classes and convert those to rules that have the minimum information necessary to discriminate between those classes. One can perform this by first changing the parameters to indicate that discriminant rules are to be produced, and then selecting this option. Although this is the typical way in which one optimizes rules, one may adjust the parameters to any setting, and then *relearn* the rule base. One may think of this procedure as improving rules with examples, but without using any input examples. In effect, the rules are being reformulated into a more useful form.

Upon selecting this option, the program goes directly to the screen in figure 28. Options after learning is complete are identical to those above.

Improving Rules with Examples in Trains	Came from: Creating Rules
<p>Example ES: trains (Enter ES containing examples or type 'r' to select from event files)</p> <p>Rules ES: trains (Enter ES containing rule base to improve or type 'r' to select from rule base)</p> <p>Output to ES: trains (Enter ES to send rules to)</p>	
Options:	
Start Learning	Return to Developing ES
Return to Creating Rules	Quit

Figure 29. Improving Rules with Examples

4) Edit Learning Parameters

There are four main parameters that affect the type of rules the learning module will produce. They are shown in figure 30 and described in greater detail below. One changes the setting of a parameter with the right and left directional keys, while moving from one parameter to the next with the up and down directional keys.

The learning parameters are:

a) *Mode for nominal output variable (default: intersecting covers)*

This parameter controls whether rules with intersecting covers or rules with disjoint covers will be produced. If intersecting covers mode is selected, then rules can overlap in the space where there are no learning events. With disjoint covers, no two rules will cover any possible event.

AgAssistant: An Experimental Expert System Builder

Editing parameters for learning		Came from: Creating rules	
Mode for nominal output variable:	intersecting covers	disjoint covers	
Scope of search: 3	(1-30)		
Produce weights: yes no			
Rule type:	Characteristic	Discriminant	Minimal complexity
	Minimal cost	User defined	
User Defined Criteria	Rank	Tolerance	
	(1-5)	(0-100 %)	
Maximize coverage of positive events not previously covered	1	30	
Minimize the number of references	2	10	
Minimize the number of selectors	3	0	
Minimize the cost of the variables			
Maximize the number of selectors			
Options:			
↑ - previous parameter	← - previous item	ESC - Other options	
↓ - next parameter	→ - next item	? - Help with parameter	

Figure 30. Editing learning parameters

b) Scope of Search (default: 3)

This parameter controls the width of the beam search that the Aq algorithm uses in its search for rules. Raising the value of this parameter may increase the quality of the generated rules; however, it will also increase the time needed by the program.

c) Produce weights (default: yes)

This parameter specifies whether the user wishes to produce weights to be associated with each selector in the created rules. If it set to no, weights will be displayed during the learning process, but will not be permanently stored with the rules base.

d) Rule type (default: characteristic)

This parameter is perhaps the most important in determining the nature of the produced rules and calls for some explanation. There are four basic types of rules. These types are shorthand for the level of generality of the rules, and the user defined criteria. When one moves from one type to another, these other parameters will change accordingly. Additionally, if one wishes to define one's own rule type, one can choose *User defined*.

Characteristic rules are those that describe every known feature of a class of objects. They are the most specific rules the program can produce and contain more information than is needed to discriminate between the given classes. Discriminant rules, on the other hand, contain only those features necessary to discriminate one class from another. Thus they are the most general descriptions of the learning events. Minimal complexity rules are somewhere in-between the two former rule types. They are neither the most general nor the most specific rules that could be produced. Minimal cost rules are those rules which favor variables that are associated with the lowest cost. (Setting the cost of a variable is described in section 6.3.4). The criteria *Minimize the cost of the variables* is ranked the highest when working with this type.

If one selects *User Defined* as a rule type, one can then define a new level of generality and set of criteria. The level of generality may range from most specific to most general. Further more, there are five criteria that one can rank. These criteria are used in evaluating competing hypotheses prior to reducing the numbers to the beam width. One does this by assigning a number from one to five with each of the criteria. Finally, one may optionally specify a tolerance for each criterion. This is a number between 0 and 100 percent which represents an uncertainty one has in applying this criterion. Thus, specifying a high tolerance for will soften the distinction between complexes of roughly similar value as evaluated by the given criterion.

6.3.3 Accessing or Creating Examples

All of the learning options in AgAssistant, with the exception of rule optimization, require examples that characterize the decision classes of the given domain. A special editor, pictured in figure 31, aids one in creating these examples.

One moves from cell to cell in this screen in the usual manner with the directional keys. Typically, one will first enter the relevant variables for the example set at the top of the screen. One then defines the type of the variable, and if it is not nominal, the system will take one to the variable definition table (described in section 6.3.4) to define the domain. One then enters the decision variable and its values on the leftmost column of the screen. For example, in figure 31, the variables are size, color, fuel and car shape. Size is a linear variable, car shape is struc-

AgAssistant: An Experimental Expert System Builder

ture, and the other are nominal. The decision variable is Direction, and its values are Northbound, Eastbound, and Southbound. Thus we are trying to find rules that describe the direction a train is moving on the basis of the values of the given variables. Once having learned those rules, it is possible to add new examples (e.g., Westbound examples) to the table, and then select the learning option of *Improving rules with examples*.

Editing Examples in train		Came from: Developing Expert System			
	DECISION	EXAMPLE			
VAR.	Direction	1. Size	2. Color	3. Fuel	4. Car shape
TYPE	nominal	linear	nominal	nominal	structured
#1	Northbound	medium	blue	oil	box
2	Northbound	small	green	gas	box
3	Eastbound	medium	red	electric	cylinder
4	Eastbound	large	purple	oil	sphere
5	Southbound	small	brown	oil	pyramid
6	Southbound	medium	brown	gas	box
7					
8					
9					
10					
Options:					
PgUp-Page up	F1-Insert row	F5-Grab column	F9-Edit domain		
PgDn-Page down	F2-Insert Column	F6-Type/Cost	F10-See values		
CTRL← Page left	F3-Grab word	F7-Jump to decision	ESC-More options		
CTRL→ Page right	F4-Grab row	F8-Jump to variable			

Figure 31. Editing examples

Options for this screen are:

F1) Insert row

Inserts a blank row between the current and next rows

F2) Insert column

Inserts a column between the current and next columns.

AgAssistant: An Experimental Expert System Builder

F3) Grab word

Highlights current word and gives the options of copying the word to the cell below, copying it throughout the row, or deleting it.

F4) Grab row

Highlights current row and gives the options of inserting a copy of the row in the next row, copying the row to another row, moving the row to another row, switching a row with another row, or deleting the row.

F5) Grab column

Highlights current column and gives the options of inserting a copy of the column in the next column, copying the column to another column, moving the column to another column, switching a column with another column, or deleting the column.

F6) Type/Cost

Toggles between displaying the variable type and the variable cost. Variable costs are in the range of 0-100. The cost of a variable indicates the relative difficulty one has in measuring its value. The learning parameters can be set to favor low cost over high cost variables.

F7) Jump to decision

Go to the left most column from anywhere in the table.

F8) Jump to variable

Go to the topmost row from anywhere in the table.

F9) Edit domain

Go to the variable definition table described in section 6.3.4.

F10) See values

See the allowed values for the given column.

ESC) More options:

Provides all the options for saving the table and moving to different screens. These are:

AgAssistant: An Experimental Expert System Builder

a) Save and Resume Edit

Save examples and resume editing them.

b) Save and Learn

Save the examples and go to learning rules.

c) Save and Developing ES

Save and to to Developing an Expert System Menu.

d) Resume Edit

Resume editing the examples.

e) Learning Menu

Go to learning rules without saving the examples.

f) Developing ES menu

Go the the Developing ES menu without saving examples.

g) Append file

Append an example set from another expert system to the current example set.

h) Import file

Import a file created with another editor. This file must be an ASCII file and conform to the structure of the example editor. That is, the first line of the file must be the variables, the second line the variable type, the third line the cost of the variables, and the following lines the examples. Spacing between cells can be arbitrary.

6.3.4 Accessing or Creating Variables

The variable table shown in figure 32 contains the definitions of the variable types and their respective domains. One will typically fill out this table first in the course of building an expert system. The table works similar to the example editor in that one moves from cell to cell with the direction keys. One difference is that one

AgAssistant: An Experimental Expert System Builder

may indent a cell if the column corresponds to a structured variable. In this example, the variable *Car_shape* is structured such that a cylinder and a sphere are considered round, while the box and pyramid are not-round. A full description of all of the variable types and their use can be found in section 6.4.

Editing Variables in train		Came from: Developing Expert System		
Variable Domains				
VAR.	1. Size	2. Color	3. Fuel	4. Car shape
TYPE	linear	nominal	nominal	structured
#1	small	blue	oil	round
2	medium	green	gas	cylinder
3	large	red	electric	sphere
4		purple		not-round
5		brown		pyramid
6				box
7				
8				
9				
10				
Options:				
PgUp - Page up	TAB - Indent value	F3 - Grab value	ESC - More options	
PgDn - Page down	BTAB - Deindent value	F4 - Grab column		
CTRL← Page left	F1 - Insert column	F5 - Type/Cost		
CTRL→ Page right	F2 - Insert value	F6 - Jump to variable		

Figure 32. Editing variables

Options for this screen are:

TAB) Indent value

If the variable is structured, then one can indicate that a value is an instance of a value representing a super class by indenting it below that value. E.g., in the menu above, cylinders and spheres are both round.

BTAB) Unindent value

Move a cell back to its former position before indenting it.

AgAssistant: An Experimental Expert System Builder

F1) Insert column

Insert a blank column to the right of the current column.

F2) Insert value

Insert a blank value immediately below the current value.

F3) Grab value

Grabs value in current cell and gives the option of deleting the value, or switching it with another value in the same column.

F4) Grab column

Grabs current column and gives the options of deleting it, switching it with another column, copying it to another column, or moving it to another column.

F5) Type/Cost

Toggles the second row between entering the variable cost and the variable type.

F6) Jump to variable

Jump to the first row from anywhere in the table.

ESC) More options:

The options that appear upon typing ESC depend on whether one went to the current screen via the example editor or the developing system menu. Options are:

a) Return to Editing Examples

Return to the example editor. The modifications made here will be saved only if one saves the examples.

b) Resume Editing Variables

As stated.

c) Save and Resume Edit

Save the current table and resume editing it.

d) *Developing ES Menu*

Return to developing an Expert System without saving the most recent modifications.

6.3.5 Accessing or Creating System Description

One chooses this option when one wants to create a new system or change the description of an existing system. As can be seen in figure 33, the system description consists of four items. They are:

Editing Description of trains		Came from: Developing an Expert System	
Expert System Domain:	Object Recognition		
(Enter 1-18 chars. or type ? to select among existing domains)			
Expert System Name:	trains		
(Enter 1-8 chars.)			
Short System Description:	Distinguishing trains by direction		
(Enter 1-50 chars.)			
Filename of Long Description:	train.txt		
(Enter 1-12 chars.)			
Options:			
Create New system and Return to Developing ES Menu		Resume entering items	
Return to ES Menu without Creating		Quit	

Figure 33. Editing System Description

1) Expert System Domain

This indicates the general category in which the system falls. Typically, systems in the same domain share variable definitions and possibly other information.

AgAssistant: An Experimental Expert System Builder

2) Expert System Name

The name of the expert system. This slot is blank if one is creating a description of <New System>. One cannot build a new expert system until it is given a name in this menu.

3) Short System Description

This description will appear on the menu for Getting Advice from an Expert System.

4) File name of Long Description

This item indicates a text file containing an in-depth description of the system. This text can be viewed by typing "?" in the menu for Getting Advice from an Expert System.

Options for this menu are:

a) Create New System and Return to Developing ES Menu

Create a new system according to the modifications made. If the name of an existing system was changed, the entire system will be copied to the new name. Returns to "Developing an Expert System" when complete.

b) Return to Es Menu with Creating

Ignore all changes made, and return to "Developing an Expert System".

6.3.6 Copying

Copy will copy the item selected from the current system to one indicated. For example, the system displays the following when copy is selected to operate on the rules in trains.

Copy rules from trains to: _____

(Enter an Expert System name or type ESC to return to Developing ES)

6.3.7 Deleting

Deleting will erase the specified item in the expert system. For example, the system displays:

Are you sure you want to delete the variables in trains (y/n)? ___

When delete is selected to operate on variables in trains.

6.3.8 Printing

Prints the selected item for the indicated rule base.

6.4 Variable Types

Variables in the AgAssistant system fall into five categories: nominal, linear, integer, numeric, and structured. The type of the variable and its domain are defined with the Variable Table (see section 6.34 for more details on this table). Below each variable type is described in more detail.

6.4.1 Nominal variables

Nominal variables have values which do not have any explicit structure. Examples of this type include boolean variables which may take the values of true or false, a variable such as color, which may take the values red, green, blue, etc., and all variables the values for which cannot be ordered in any reasonable fashion.

6.4.2 Linear variables

Linear variables are those with values that can be ordered in some fashion. Examples of this type are variables such as size, taking on the values of small, medium, and large, and age, which could consist of the ordered domain infant, child, adolescent, young-adult, middle-age, and old-age. Linear variables can appear in rules with the "to" separator. For example, a condition in a rule might be that "Age is child to young-adult". The system understands this to include the value "adolescent"

6.4.3 Integer variables

Integer variables, pre-defined as a convenience to the user, are linear variables with a range that extends

AgAssistant: An Experimental Expert System Builder

from the integer 0 to the integer 50. Thus, a valid condition involving an integer variable might be, "No of flowers is 29 to 33".

6.4.4 Numeric variables

Numeric variables are defined as a range between two real numbers. Thus valid numeric variable would be pH, with a range from 0.0 to 14.0. pH could then take any real value within this range, for example, 5.6128. Numeric variables, and only numeric variables, can appear in arithmetic expressions in rules. An example of a valid arithmetic expression involving numeric variables is "(pH * solvability) > (5.8 * density)". Refer to section 3.3 for a description of the syntax of arithmetic expressions. For the purposes of induction, a numeric range is split into 8 categories of equal length. Thus, pH would be split into 0.0-1.75, 1.75-3.5, etc.

6.4.5 Structured variables

Structured variables have values that be ordered in a hierarchical manner. An example of the structure of the variable shape appears below.



Figure 34. Structure for the variable shape

The rule learning module will attempt to "climb the tree" in producing descriptive rules. For example, in the above figure, if two examples for one decision class have the values for shape of square and rectangle, then the condition produced will not be "Shape is square or rectangle", but "Shape is four-sided". Additionally, the advisory system makes use of the variable structure in its attempt to satisfy rule conditions. For example, if one answers the question "What shape does the object have" with the value "ellipse", then the system understands that "Shape is round" is also satisfied.

6.5 Rule Compiler Error Messages

Rule compiler error messages are divided into those that are fatal, and do not result in a working advisory system, and warnings, which inform the user of certain problems during the parse, but will result in a new system. The system will attempt to continue even if it encounters a fatal error, in order to inform the user of other possible problems with the rules. If possible, the system gives the line where the error occurred.

Fatal Errors

1) *Action not found. Skipping to next rule.*

An action in the form of <decision variable> <relation> <value> was not found at the top of the rule.

2) *Number not found in above selector. Skipping to next selector.*

A number was not found before the selector. Selector is ignored.

3) *"." not found after number. Skipping to next selector.*

A period was not found after the number of the selector.

4) *Variable not found in above selector. Skipping to next selector.*

A variable was not found after the number.

5) *Relation not found in above selector. Skipping to next selector.*

A relation was not found following the variable.

6) *Value not found in above selector. Skipping to next selector.*

A value was not found following the relation.

7) *"or" or "to" not found between values. Skipping to next selector.*

All values must be in a disjunctive set or in a linear range.

8) *Final Value missing. Skipping to next selector.*

A value was expected after "to" in a range description.

AgAssistant: An Experimental Expert System Builder

9) *"," or "." not found. Skipping to next selector.*

Each selector must be followed by a comma or a period. A period tells the parser that it is the last selector in the rule.

10) *Weight no in 0-100 range.*

Weight was not an integer, or not in the proper range.

11) *"to" construction used with non-linear variable. Please declare <variable> as a linear variable.*

The variable was not declared as linear. Thus it cannot be associated with a range.

12) *Linear value <value> was not found in domain.*

All linear values must be predeclared in order that the compiler can construct the correct range.

13) *Variable <variable> used in arithmetic expression but not declared as numeric.*

All variables appearing in arithmetic expressions must be numeric.

14) *Unbalanced parentheses in arithmetic expression.*

The number of left parentheses must match the number of left parentheses in an arithmetic expression.

Warnings

1) *Variable too long. Truncating to 12 characters.*

In the current system, all variables are limited to 12 characters. It is truncated if it is too long.

2) *Relation too long. Truncating to 12 characters.*

Relation must be 12 or less characters.

3) *Value too long. Truncating to 12 characters.*

Value must be 12 or less characters.

4) *<variable> not declared. Assuming nominal type.*

The variable was not found in the variable table. It is assumed that it is nominal.

7. Rules for WEEDER

Does not apply:

Flow_exst

yes Auricle Blade_width

no Florets Flower Awns Disarticu Glumes

Weed is Bahiagrass if: cl

- | | |
|---------------------------|----|
| 1. Habit is rhiz_stolon, | 85 |
| 2. Flower is raceme, | 70 |
| 3. Florets is 1, | 60 |
| 4. Ligule is truncate, | 55 |
| 5. Blade_width is coarse, | 50 |
| 6. Sheath is compressed, | 35 |
| 7. Collar is broad, | 35 |
| 8. Disarticu is below, | 35 |
| 9. Glumes are shorter. | 20 |

Weed is Alkaligrass if: cl

- | | |
|--------------------------|----|
| 1. Florets is 2, | 80 |
| 2. Flower is panicle, | 35 |
| 3. Habit is rhiz_stolon, | 85 |
| 4. Disarticu is above, | 35 |
| 5. Glumes are shorter. | 10 |

Weed is Hardgrass if: cl

- | | |
|------------------------|----|
| 1. Flower is raceme, | 70 |
| 2. Florets is 3, | 65 |
| 3. Habit is bunch, | 45 |
| 4. Disarticu is below, | 15 |
| 5. Glumes are shorter. | 10 |

Weed is Smutgrass if: cl

- | | |
|------------------------|----|
| 1. Florets is 1, | 80 |
| 2. Flower is panicle, | 70 |
| 3. Ligule is ciliate, | 70 |
| 4. Habit is bunch, | 65 |
| 5. Disarticu is above, | 35 |
| 6. Glumes are shorter, | 25 |
| 7. Awns are absent. | 10 |

Weed is Fall_panicum if: cl

- | | |
|------------------------|----|
| 1. Florets is 1, | 80 |
| 2. Flower is panicle, | 80 |
| 3. Ligule is ciliate, | 70 |
| 4. Habit is bunch, | 65 |
| 5. Glumes are shorter, | 30 |
| 6. Disarticu is below. | 20 |

Weed is Torpedograss if: cl

- | | |
|---------------------------|----|
| 1. Habit is rhizome, | 80 |
| 2. Ligule is ciliate, | 65 |
| 3. Flower is panicle, | 60 |
| 4. Florets is 1, | 55 |
| 5. Sheath is compressed, | 50 |
| 6. Blade_width is coarse, | 45 |
| 7. Glumes are shorter, | 25 |
| 8. Disarticu is below. | 15 |

Weed is Anl_dropseed if: cl

- | | |
|---------------------------|----|
| 1. Ligule is round, | 70 |
| 2. Sheath is round, | 60 |
| 3. Florets is 1, | 60 |
| 4. Flower is panicle, | 55 |
| 5. Blade_width is medium, | 50 |
| 6. Habit is bunch, | 35 |
| 7. Glumes are shorter, | 20 |
| 8. Disarticu is above. | 15 |

Weed is Frif_Paspalum if: cl

- | | |
|---------------------------|----|
| 1. Flower is raceme, | 85 |
| 2. Ligule is round, | 80 |
| 3. Florets is 1, | 70 |
| 4. Blade_width is medium, | 50 |
| 5. Sheath is compressed, | 40 |
| 6. Habit is bunch, | 40 |
| 7. Glumes are shorter, | 35 |
| 8. Disarticu is below. | 35 |

AgAssistant: An Experimental Expert System Builder

Weed is Cogongrass if:	cl	Weed is Orchardgrass if:	cl
1. Ligule is truncate,	80	1. Ligule is toothed,	70
2. Sheath is round,	75	2. Florets is 4,	65
3. Habit is rhizome,	75	3. Sheath is closed,	65
4. Flower is spike,	70	4. Vernation is folded,	60
5. Blade_width is coarse,	55	5. Blade_width is coarse,	55
6. Florets is 1,	50	6. Habit is bunch,	50
7. Glumes are shorter,	20	7. Collar is broad,	45
8. Disarticu is below.	20	8. Disarticu is above,	40
		9. Flower is panicle,	30
		10. Glumes are shorter.	15
Weed is Anl_Bluegras if:	cl	Weed is Goosegrass if:	cl
1. Ligule is acute,	90	1. Ligule is toothed,	75
2. Florets is 3 to 6,	80	2. Vernation is folded,	65
3. Collar is narrow,	60	3. Flower is raceme,	65
4. Flower is panicle,	60	4. Blade_width is medium,	50
5. Blade_width is fine to medium,	55	5. Florets is 6,	50
6. Vernation is folded,	50	6. Habit is bunch,	45
7. Habit is bunch or stolon,	35	7. Collar is broad,	40
8. Sheath is compressed,	25	8. Sheath is compressed,	25
9. Glumes are shorter,	15	9. Disarticu is above,	25
10. Disarticu is above.	15	10. Glumes are shorter.	15
Weed is Kikuyugrass if:	cl	Weed is Per_ryegrass if:	cl
1. Habit is rhiz_stolon,	90	1. Ligule is round,	85
2. Flower is panicle,	60	2. Auricle is short,	80
3. Vernation is folded,	55	3. Florets is 6 to 10,	80
4. Collar is narrow,	40	4. Flower is spike,	75
5. Florets is 2,	40	5. Vernation is folded,	50
6. Blade_width is medium,	30	6. Habit is bunch,	40
7. Ligule is ciliate,	25	7. Collar is broad or divided,	35
8. Disarticu is below,	25	8. Sheath is compressed,	30
9. Glumes are shorter.	15	9. Blade_width is fine to medium,	30
		10. Disarticu is above,	35
		11. Glumes are shorter.	25
Weed is Bermudagrass if:	cl	Weed is Dallisgrass if:	cl
1. Habit is rhiz_stolon,	95	1. Ligule is acuminate,	90
2. Vernation is folded,	70	2. Flower is raceme,	75
3. Ligule is ciliate,	70	3. Habit is bunch or rhizome,	75
4. Disarticu is above,	65	4. Blade_width is coarse,	55
5. Flower is spike,	60	5. Sheath is compressed,	45
6. Blade_width is fine or medium,	50	6. Collar is broad,	30
7. Sheath is compressed,	25	7. Vernation is rolled,	30
8. Collar is narrow,	25	8. Florets is 1,	30
9. Florets is 1,	25	9. Disarticu is below,	30
10. Glumes are shorter.	15	10. Glumes are shorter.	25
		Weed is Yellw_Fxtail if:	cl

AgAssistant: An Experimental Expert System Builder

1. Florets is 2,	75	1. Habit is rhiz_stolon,	80
2. Ligule is ciliate,	65	2. Glumes are longer,	80
3. Flower is spike,	65	3. Awns are present,	75
4. Blade_width is coarse,	50	4. Flower is spike,	70
5. Collar is broad,	40	5. Sheath is round,	70
6. Sheath is compressed,	30	6. Ligule is ciliate,	60
7. Habit is bunch,	30	7. Florets is 1,	55
8. Vernation is rolled,	30	8. Blade_width is medium,	50
9. Glumes are shorter,	30	9. Collar is broad,	50
10. Disarticu is above.	25	10. Disarticu is below,	45
		11. Vernation is rolled.	35
Weed is Field_sandbr if:	cl	Weed is Junglerice if:	cl
1. Florets is bur,	90	1. Ligule is none,	95
2. Ligule is ciliate,	55	2. Flower is raceme,	75
3. Collar is broad,	35	3. Sheath is compressed,	55
4. Sheath is compressed,	30	4. Blade_width is medium,	55
5. Blade_width is medium,	30	5. Habit is bunch,	50
6. Disarticu is below,	30	6. Collar is broad,	40
7. Habit is bunch,	30	7. Disarticu is below,	40
8. Vernation is rolled.	25	8. Vernation is rolled,	35
		9. Florets is 1,	35
Weed is Witchgrass if:	cl	10. Glumes are shorter.	35
1. Ligule is ciliate,	65	Weed is Barnyardgrass if:	cl
2. Flower is panicle,	60	1. Ligule is none,	95
3. Collar is narrow,	50	2. Flower is panicle,	80
4. Blade_width is medium,	45	3. Blade_width is coarse,	75
5. Habit is bunch,	45	4. Vernation is rolled,	50
6. Sheath is compressed,	40	5. Sheath is compressed,	50
7. Vernation is rolled,	40	6. Habit is bunch,	50
8. Glumes are shorter,	35	7. Collar is broad,	50
9. Florets is 1,	30	8. Florets is 1,	50
10. Disarticu is below.	30	9. Glumes are shorter,	35
		10. Disarticu is below.	35
Weed is Stinkgrass if:	cl	Weed is Nimblewill if:	cl
1. Florets is 10 to 12,	85	1. Habit is stolon,	75
2. Blade_width is fine,	70	2. Awns are present,	70
3. Flower is panicle,	60	3. Blade_width is fine,	65
4. Habit is bunch,	55	4. Ligule is round,	60
5. Collar is narrow or broad,	50	5. Flower is panicle,	60
6. Ligule is ciliate,	50	6. Sheath is compressed,	40
7. Disarticu is above,	35	7. Vernation is rolled,	35
8. Sheath is compressed,	30	8. Collar is broad,	35
9. Vernation is rolled,	30	9. Florets is 1,	35
10. Glumes are shorter.	25	10. Glumes are shorter,	35
		11. Disarticu is above.	25
Weed is Zoysiagrass if:	cl	Weed is Com_Velvtgrs if:	cl

AgAssistant: An Experimental Expert System Builder

1. Ligule is round,	75	1. Ligule is toothed or acute,	65
2. Glumes are longer,	75	2. Blade_width is coarse,	60
3. Awns are present,	75	3. Flower is spike,	60
4. Florets is 2,	65	4. Sheath is compressed,	50
5. Blade_width is fine,	55	5. Habit is bunch,	40
6. Collar is narrow,	50	6. Disarticu is below,	40
7. Flower is panicle,	45	7. Collar is broad,	35
8. Disarticu is below,	40	8. Florets is 1,	35
9. Habit is bunch,	35	9. Vernation is rolled,	35
10. Sheath is compressed,	35	10. Glumes are shorter.	20
11. Vernation is rolled.	35		
Weed is Johnsongrass if:	cl	Weed is Redtop if:	cl
1. Ligule is round,	65	1. Habit is rhizome,	75
2. Blade_width is coarse,	65	2. Ligule is toothed or acute,	60
3. Habit is rhizome,	60	3. Glumes are longer,	60
4. Awns are absent or present,	60	4. Florets is 1,	50
5. Sheath is round,	50	5. Flower is panicle,	50
6. Flower is panicle,	50	6. Sheath is round,	45
7. Collar is broad,	45	7. Disarticu is above,	40
8. Disarticu is below,	30	8. Blade_width is medium or coarse,	35
9. Vernation is rolled,	30	9. Collar is broad,	35
10. Florets is 1,	30	10. Vernation is rolled.	35
11. Glumes are shorter.	30		
Weed is Bentgrass if:	cl	Weed is Rescuegrass if:	cl
1. Ligule is round,	65	1. Sheath is closed,	80
2. Sheath is round,	65	2. Ligule is toothed,	70
3. Glumes are longer,	65	3. Florets is 6 to 12,	70
4. Habit is stolon,	60	4. Habit is bunch,	60
5. Disarticu is above,	55	5. Collar is broad,	55
6. Collar is narrow,	50	6. Blade_width is coarse,	45
7. Florets is 1,	45	7. Vernation is rolled,	35
8. Flower is panicle,	45	8. Disarticu is above,	35
9. Blade_width is fine,	35	9. Flower is panicle,	35
10. Vernation is rolled.	30	10. Awns are absent or present,	25
		11. Glumes are shorter.	20
Weed is Ken_Bluegrass if:	cl	Weed is Fxtail_Barly if:	cl
1. Vernation is folded,	65	1. Ligule is toothed,	70
2. Habit is rhizome,	60	2. Awns are present,	70
3. Ligule is truncate,	55	3. Flower is spike,	65
4. Florets is 3 to 5,	55	4. Blade_width is medium,	55
5. Flower is panicle,	45	5. Habit is bunch,	50
6. Collar is divided,	40	6. Collar is narrow,	50
7. Disarticu is above,	35	7. Vernation is rolled,	45
8. Blade_width is fine to medium,	30	8. Sheath is compressed,	40
9. Glumes are shorter.	25	9. Florets is 1,	40
Weed is Lg_Crabgrass if:	cl	10. Disarticu is above,	30
		11. Glumes are shorter.	30
		Weed is Wild_Oats if:	cl

AgAssistant: An Experimental Expert System Builder

1. Awns are bifid,	80	1. Ligule is truncate,	65
2. Ligule is toothed,	70	2. Sheath is round,	65
3. Sheath is round,	70	3. Florets is 6 to 8,	60
4. Florets is 2 or 3,	70	4. Habit is bunch,	50
5. Glumes are longer,	70	5. Collar is broad,	40
6. Habit is bunch,	50	6. Blade_width is coarse,	40
7. Flower is panicle,	50	7. Vernation is rolled,	35
8. Collar is broad,	40	8. Flower is panicle,	35
9. Blade_width is coarse,	35	9. Disarticu is above,	35
10. Vernation is rolled,	20	10. Glumes are shorter.	30
11. Disarticu is above.	20		
Weed is Downy_Brome if:	cl	Weed is Fine_Fescue if:	cl
1. Awns are bifid,	80	1. Blade_width is fine,	65
1. Ligule is toothed,	75	2. Awns are present,	65
2. Sheath is closed,	75	3. Florets is 4 to 6,	55
4. Habit is bunch,	60	4. Vernation is folded,	45
2. Florets is 3 to 6,	60	5. Ligule is truncate or round,	45
3. Flower is panicle,	55	6. Sheath is round,	40
5. Collar is narrow,	50	7. Habit is bunch or rhizome,	40
3. Blade_width is medium,	40	8. Flower is panicle,	35
6. Vernation is rolled,	40	9. Disarticu is above,	30
4. Disarticu is above,	30	10. Glumes are shorter.	25
5. Glumes are shorter.	30		
Weed is Timothy if:	cl	Weed is Smooth_Brome if:	cl
1. Ligule is toothed,	70	1. Sheath is closed,	75
2. Collar is narrow,	70	2. Habit is rhizome,	75
3. Awns are present,	70	3. Ligule is truncate,	65
4. Sheath is round,	65	4. Florets is 8,	60
5. Flower is spike,	65	5. Blade_width is coarse,	50
6. Habit is bunch,	50	6. Collar is broad,	30
7. Disarticu is above,	45	7. Vernation is rolled,	30
8. Blade_width is coarse,	35	8. Flower is panicle,	30
9. Vernation is rolled,	35	9. Disarticu is above,	30
10. Florets is 1,	30	10. Glumes are shorter.	30
11. Glumes are shorter.	25		
Weed is Sm_Crabgrass if:	cl	Weed is Itln_Ryegrss if:	cl
1. Ligule is truncate,	75	1. Florets is 10 to 20,	85
2. Habit is bunch,	70	2. Auricle is claw_like,	85
3. Flower is spike,	70	3. Awns are present,	70
4. Disarticu is below,	60	4. Flower is spike,	65
5. Collar is broad,	55	5. Ligule is round,	60
6. Sheath is compressed,	45	6. Sheath is round,	60
7. Blade_width is coarse,	30	7. Collar is broad,	50
8. Vernation is rolled,	30	8. Blade_width is coarse,	45
9. Florets is 1,	30	9. Habit is bunch,	45
10. Glumes are shorter.	25	10. Vernation is rolled,	35
		11. Disarticu is above,	30
		12. Glumes are shorter.	30
Weed is Tall_Fescue if:	cl	Weed is Quackgrass if:	cl

AgAssistant: An Experimental Expert System Builder

1. Auricle is claw_like,	80
2. Ligule is truncate,	80
3. Habit is rhizome,	75
4. Awns are present,	60
5. Flower is spike,	60
6. Florets is 4 to 6,	55
7. Sheath is round,	55
8. Blade_width is coarse,	45
9. Collar is broad,	45
10. Vernation is rolled,	35
11. Disarticu is above,	30
12. Glumes are shorter.	30

7.1 Modified WEEDER rules modifications shown in BOLD

Weed is Per_ryegrass if:	cl
1. Ligule is round or truncate,	25
2. Auricle is short,	80
3. Florets is 6 to 10,	80
4. Flower is spike,	75
5. Vernation is folded,	50
6. Habit is bunch,	40
7. Collar is broad or divided,	75
8. Sheath is compressed,	70
9. Blade_width is fine to medium,	70
10. Disarticu is above,	35
11. Glumes are shorter.	25

Weed is Zoysiagrass if:	cl
1. Habit is rhiz_stolon or rhizome,	80
2. Glumes are longer,	80
3. Awns are present,	75
4. Flower is spike,	70
5. Sheath is round,	70
6. Ligule is ciliate,	60
7. Florets is 1,	55
8. Blade_width is fine to medium,	50
9. Collar is broad,	70
10. Disarticu is below,	45
11. Vernation is rolled.	75

Weed is Bentgrass if:	cl
1. Ligule is round or toothed,	65
2. Sheath is round,	65
3. Glumes are longer,	65
4. Habit is stolon,	40
5. Disarticu is above,	55
6. Collar is narrow,	70
7. Florets is 1,	45
8. Flower is panicle,	45
9. Blade_width is fine,	70
10. Vernation is rolled.	70

Weed is Lg_Crabgrass if:	cl
1. Ligule is toothed or acute,	65
2. Blade_width is medium,	60
3. Flower is spike,	60
4. Sheath is compressed,	50
5. Habit is bunch,	60
6. Disarticu is below,	40
7. Collar is broad or divided,	35
8. Florets is 1,	35
9. Vernation is rolled,	75
10. Glumes are shorter.	20

8. References

1. Dietterich, T. and R. S. Michalski. 1983. A Comparative Review of Selected Methods for Learning from Examples. Chapter in the book, *MACHINE LEARNING: An Artificial Intelligence Approach*, TIOGA Publishing Company, Palo Alto, R. S. Michalski, J. Carbonell and T. Mitchell (Eds.), 1983, pp. 41-81.
2. Michalski, R. S. 1973. AQVAL/1 Computer Implementation of a Variable Valued Logic System VL1 and Examples of its Application to Pattern Recognition. *Proceedings of the First International Joint Conference on Pattern Recognition.*, Washington, D. C. pp. 317.
3. Michalski, R. S. 1975. Variable Valued Logic and its Application to Pattern Recognition and Machine Learning. in *Computer Science and Multiple Valued Logic: Theory and Applications*, D. C. Rine (ed.), North-Holland. pp. 506-534.
4. Michalski, R. S. 1983. A Theory and Methodology of Inductive Learning. Chapter in the book, *MACHINE LEARNING: An Artificial Intelligence Approach*, TIOGA Publishing Company, Palo Alto, R. S. Michalski, J. Carbonell and T. Mitchell (Eds.), 1983, pp. 83-134.
5. Michalski R. S. 1985. Knowledge Repair Mechanisms: Evolution vs Revolution, ISG 8514, UTUCDCSF85946, Department of Computer Science, University of Illinois, Urbana, IL, July 1985, and the *Proceedings of the Third International Machine Learning Workshop, Skytop, Rutgers University*, June 1985.
6. Michalski, R. S and A. B. Baskin. 1983. Integrating Multiple Knowledge Representations and Learning Capabilities in an Expert System: The ADVISE System. *Proceedings of the 8th IJCAI*. Karlsruhe, West Germany, August 8-12, 1983. pp. 69-79.
7. Michalski, R. S, and J. B. Larson. INCREMENTAL GENERATION OF VL1 HYPOTHESIS: the underlying methodology and the description of program AQ11. ISG 835, UTUCDCSF83905, Department of Computer Science, University of Illinois, 1983.
8. Michalski, R. S, and R. L. Chilausky. 1980a. Knowledge Acquisition by Encoding Expert Rules versus Computer Induction From Examples: A Case Study Involving Soybean Pathology. *International Journal for ManMachine Studies*. No. 12, 1980, pp. 63-87.
9. Michalski, R. S, and R. L. Chilausky. 1980b. Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis. *International Journal of Policy Analysis and Information Systems*, Vol. 4, No. 2, 1980.
10. Michalski, R. S., J. H. Davis, V. S. Bisht and J. B. Sinclair. 1982. PLANT/ds: An Expert Consulting System for the Diagnosis of Soybean Diseases Accepted for publication in *PLANT DISEASES* and Proc. of the 1982 European Conference on Artificial Intelligence, Orsay, France, July 12-14, 1982, pp. 133-138
11. Morse, L. E. 1971. Specimen identification and key construction with time-sharing computers. *Taxon*, 20(1):269-82.
12. Reinke, R. E. 1984. Knowledge Acquisition and Refinement Tools for the ADVISE MetaExpert System. M. S. Thesis ISG 844, UTUCDCSF84921, Department of Computer Science, University of Illinois, July, 1984.

AgAssistant: An Experimental Expert System Builder

13. Shurtleff, M. C., T. W. Fermanian, and R. Randell. Controlling Turfgrass Pests Prentice-Hall, Inc. Englewood Cliffs, N.J. 462 pp.
14. Stepp, R. 1983. A Description and User's Guide for CLUSTER/2, A Program for Conjunctive Conceptual Clustering, Report No. UIUCDCSR831084, Department of Computer Science, University of Illinois, Urbana, IL, November 1983.

BIBLIOGRAPHIC DATA SHEET		1. Report No. UIUCDCS-F-87-978	2.	3. Recipient's Accession No.
4. Title and Subtitle				5. Report Date
AgAssistant: An Experimental Expert System Builder for Agricultural Applications				6.
7. Author(s) Bruce Katz, Thomas W. Fermanian, Ryszard S. Michalski				8. Performing Organization Repr. No.
9. Performing Organization Name and Address Department of Computer Science and Department of Horticulture University of Illinois at Urbana-Champaign				10. Project/Task/Work Unit No.
				11. Contract/Grant No.
12. Sponsoring Organization Name and Address International Intelligent Systems, Inc. The University of Illinois Research Board Project # ILLU-65-0357, of the Agricultural Experiment Station College of Agriculture, University of Illinois Urbana-Champaign				13. Type of Report & Period Covered
				14.
15. Supplementary Notes				
16. Abstracts				
<p>AgAssistant is a comprehensive expert system builder for IBM PC and compatible computers in the area of agriculture. The inferencing mechanism was specifically designed to combine levels of uncertainty commonly found in agricultural domains. It is both an enhancement and an extension of an earlier expert system for the IBM PC, PLANT/ds, which was concerned with the diagnosis of soybean diseases common in Illinois. Unlike PLANT/ds, in which all modifications of the system take place on a VAX minicomputer, AgAssistant provides a set of tools for system modification and development directly on the PC. The work presented here is also based to a large extent on the ADVISE Meta-Expert System.</p> <p>AgAssistant is also more than an expert system builder. It provides a set of tools for the development of a Knowledge base by example of expert knowledge (learning by example). These learning tools may be applied to other agricultural problems where the classification or grouping of non-numeric data is useful.</p> <p>Among the many novel features incorporated into AgAssistant are:</p> <ul style="list-style-type: none"> • Multiple means of creating and refining knowledge. <p>AgAssistant can use rules developed directly from experts or acquired through inductive inference in the same format.</p> <ul style="list-style-type: none"> • Probabilistic inference can be handled. <p>Of great use in agriculture, where the vagaries of nature can make identification or diagnosis a probabilistic matter.</p> <ul style="list-style-type: none"> • The system is PC-based. <p>This allows wide dissemination of the program to farmers and others in need who are unlikely to have access to larger systems.</p> <ul style="list-style-type: none"> • Menu-driven screens. <p>The novice user can quickly come up to speed in building experts systems.</p>				
17. Key Words:				
<p>AgAssistant expert system builder PLANT/ds expert knowledge inductive inference probabilistic inference PC-based</p>				
17a. Descriptions				
17b. Identifiers/Open-Ended Terms				
17c. COSATI Field/Group				
17e. COSATI Field/Group				
18. Availability Statement			19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 72
			20. Security Class (This Page) UNCLASSIFIED	22. Price

