QUANTITATIVE EMPIRICAL LEARNING: AN ANALYSIS AND METHODOLOGY

BY

BRIAN CARL FALKENHAINER

B.S., University of Santa Clara, 1982

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1985

Urbana, Illinois

August 1985

This research was supported in part by the National Science Foundation under grant NSF DCR 84-06801 and in part by the Office of Naval Research under grant N00014-82-K-0186.

For Ernest Jungo, my grandad.
A wise and intelligent man to whom I owe a debt which could never be repaid.

ACKNOWLEDGEMENTS

I wish to thank my thesis advisor, Professor Ryszard Michalski, for his guidance and support. His suggestions early on in the project were most helpful in getting me started.

I would also like to thank Professor Ken Forbus for his helpful comments and for spending too much time proof reading my papers. His contageous enthusiasm for AI helped reignite my interest in this project at a time when it was waning.

I wish to express my gratitude to the other members of ISG for providing a cohesive environment in which to work. Tony Nowicki, Bob Stepp, and Carl Uhrik have spent much time maintaining the AI Laboratory's network of Sun Workstations. I must thank Tony Nowicki and Jeff Becker for their friendship, gripe sessions, proof reading, and contributing useful ideas to this thesis. Without Jeff introducing me to the chemistry data, helping me to "borrow" his A^q LISP program, and the use of his Franz Lisp Macro Package, progress would have been much slower.

Thanks also go to Dr. Ted Brown and Mike Hankel of the UI Chemistry department for providing useful data with which to test my program.

The talks I have had with Nayel El-Shafei at MIT have been most beneficial by forcing me to understand better what I was doing and by his pointing out possible alternative approaches.

Closer to home, I am most grateful to my parents and grandparents for their support, encouragement, and faith in me throughout the years. They are always only a phone call away. To Susan, I owe my love. I could never have come to Illinois were it not for her.

This work was supported in part by the National Science Foundation under grant NSF DCR 84-06801 and by the Office of Naval Research under grant N00014-82-K-0186.

TABLE OF CONTENTS

1. INTRODUCTION

The field of Artificial Intelligence has many facets. In recent years, the importance of enabling AI programs to learn has been stressed due to the limitations of current AI systems [Michalski, 1985]. Improving performance means increasing the knowledge available to the program, but usually this requires tedious manual insertion. Programs lacking the ability to learn cannot learn from their mistakes or experiences, cannot adjust to their surroundings, and certainly cannot be expected to learn or discover something which is new to the user as well. While it cannot be said that the growing field of machine learning has solved all these problems, advances have been many and the field is quickly maturing. This thesis addresses the task of enabling machines not just to learn, but to make new discoveries. Specifically, *quantitative empirical learning*, an important subproblem of scientific discovery, is examined.

Quantitative empirical learning is the discovery of a mathematical equation or equations which hold for an observed set of numerical data. For example, given a list of different box dimensions consisting of height, depth, width, and volume, it will always be true that the volume will equal the height times the depth times the width. Computer programs possessing the capability to discover such facts would be of great service. In the extreme, entirely new laws of physics or chemistry might be uncovered by the computer. More realistically, such a learning program could be used as a laboratory aid or as part of a larger learning system tied into some form of expert system. Given a situation where large amounts of numeric data must be analyzed, a program of this type might be able to uncover relationships in the data in situations where humans would be overwhelmed by the quantity and complexity of the data.

How then does one go about discovering mathematical relationships for some observed data? As an example, consider the data presented in Figure 1. A simple relation exists for this data, but it may not be obvious to someone examining these measurements. We are even told that this is "ideal" data, meaning that all of the variables are included in the hidden relation, all examples (rows) obey the relation, and there is absolutely no noise. Were the variable names changed to read Mass, Force, and

x	y	z
5.0	49.0	9.80
5.0	98.0	19.60
8.0	76.0	9.50
8.0	78.4	9.80
8.0	98.0	12.25
10.5	76.0	7.24
10.5	102.9	9.80

Figure 1. "Ideal" Empirical Data for Equation Learning

Acceleration rather than x, y, and z, one might soon recognize the fact that $y = xz$, but this would call upon our semantic interpretation of the words Force, Mass, and Acceleration and our prior knowledge that $F = MA$. Given that, we would have been more likely to predict the observations rather than learn from them. Quantitative empirical learning addresses the problem of discovering a relation which is not previously known.

This thesis presents an approach to quantitative learning and the theory behind it. The methods discussed have all been implemented in Franz Lisp in a program called ABACUS which runs on a Sun Microsystems Workstation. Chapter 2 discusses quantitative empirical learning in greater detail, presents some of the issues involved, discusses previous work, and introduces the approach taken in the ABACUS system. In Chapter 3 the theory behind ABACUS's equation discovery is examined while Chapter 4 introduces the idea of domain constraints. Chapter 5 analyzes a variety of "discoveries" which ABACUS has made. Finally, Chapter 6 evaluates the theory presented and indicates directions for future research. The interested reader may also want to inspect Appendix A for a discussion of an earlier ABACUS implementation.

Numerous examples are used in the following chapters for purposes of explanation and as examples of what ABACUS can discover. All examples shown have actually been tested on the implemented program and the results achieved match those either stated or implied.

2. ABACUS APPROACH TO QUANTITATIVE LEARNING

A simple example should help to clarify the types of problems addressed by quantitative learning. Suppose one is given a list of observed values for variables x and y which correspond to the graph of Figure 2. From the graphical representation, one should expect to find two different formulas that describe the graph. One formula would apply to the curve while the other would represent the equation of the line. Finding the equation of a line using two points on the line is quite simple and is taught in early algebra courses. But what of the curve? Methods to find the equation of a curve are not as well known. For more complicated curves relating numerous variables there is no simple method. Suppose one tries combining pairs of variables according to noticeable trends in the data. For example, in the case of the curve the values for y go up and down when the values of x go up and down. Since equations of the form

$$\frac{a}{b} = \text{Constant}$$

also have this property, $\frac{x}{y}$ might be a possible solution to the problem. In comparing the values of the graph to those calculated for $\frac{x}{y}$, it is seen that this is not the final solution. It can also be seen,

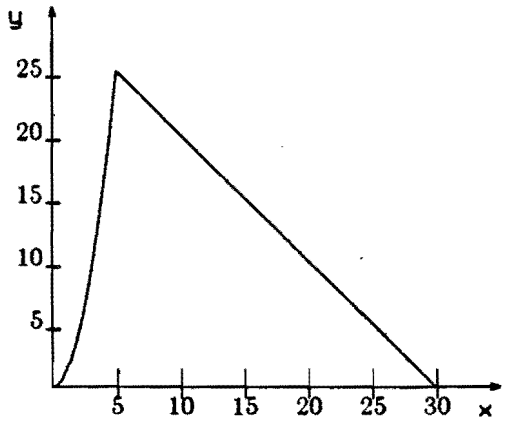


Figure 2. Equation Discovery Applied to a Graph

however, that the values for $\frac{x}{y}$ go down when x rises. Equations of the form

$$ab = \text{Constant}$$

possess this property and so one might then try the equation $x(\frac{x}{y}) = \text{Constant}$ to describe the curve. An examination of the values for this equation reveals that $y = x^2$ does in fact describe the curve. For this graph then, one can find two equations to describe the two disjoint sets of points (the curve and the line).

This thesis presents techniques which try to make no assumptions and require minimal information from the user. It is believed that while additional information from a human observer would simplify the problem, these simplifications would compromise the value of the work. Similarly, various constraints could be placed on the behavior of the data, but this again diminishes the accomplishment.

What is presented here is a program called ABACUS which implements a quantitative discovery system that tries to adhere to these objectives. The result is an approach which makes almost no assumptions about the data it is working on and thus is more general yet less powerful than one which possesses greater knowledge. The input data to ABACUS consists of variable declarations and a list of observed events. Only a variable's name, type (numeric or symbolic), and units (meters^2, etc) are specified and no designation of dependent versus independent is made. Any number of irrelevant variables may be present. The events may represent different equations and a different set of variables may apply for each equation. Thus a variable may be relevant for part of the data and irrelevant for other parts. The operators in discovered equations are limited to addition, subtraction, multiplication, and division, although the user may introduce entities such as log(x) easily through the interface module. The goal of this thesis is to investigate how far the limits of quantitative empirical learning may be stretched without the aid of limiting restrictions. This involves investigating both the techniques by which equations can be discovered and discerning when the equations can be used.

2.1. Issues

The problem of quantitative empirical learning involves many issues. One needs to decide what assumptions to make and how to make them. Many of the more useful assumptions may in turn compromise the value of what is finally achieved. This section shall present and discuss these issues and state how each is addressed by the current work.

2.1.1. Irrelevant Variables and Irrelevant Events

One of the first decisions which must be made is whether to allow irrelevant variables and irrelevant events in the observed data. Not only is the issue important in its own right, but the decisions made here will affect the handling of many of the other issues.

Assuming that all of the observed attributes must be present in any equation discovered would greatly simplify the problem. It provides useful search heuristics by stating that any variable not yet present in an incomplete equation must eventually be included. Similarly, this assumption would allow us to decisively determine whether or not the search has completed. However, this assumption greatly limits the usefulness of any discovery system based on it. In a laboratory environment one often does not know which variables are important and which are not. Limiting data to allow only relevant variables would appear to limit the system to contrived, man-made examples only. Therefore, no such assumption is made in this work.

We could also assume that all observed events are described by a single equation. While requiring all observations to be pertinent to a single relation is not quite as limiting, it would still prevent the learning system from discovering several equations (as in Figure 2) rather than one. ABACUS assumes that multiple relations may exist in the data, but all the relations must hold for disjoint subsets of the observed events. In addition, no assumption is made that any relations exist or that all events must necessarily represent some relation.

2.1.2. Finding Relations

The process of equation creation is naturally affected by and likewise affects the various assumptions to be made. Previous approaches, [Langley, 1981a; Falkenhainer, 1984], have generally agreed that the problem must be attacked as if it were "two-dimensional". By two-dimensional we mean that the relation xyz would be discovered by first forming a new variable xy (or some other pair) and then combining this variable with z.[1] Traditional curve fitting approaches have a long history and are well understood [Langley, 1983a], but the multiple equation assumption rules them out because events may not adhere to a single curve and thus may appear as noise to any curve fitting procedure. One might require that the user provide sample formulas (formula templates) so that the system need merely consider fitting each to the data in turn. Unfortunately, the multiple equation assumption again interferes by rendering "degree of fit" an unacceptable criterion. ABACUS avoids these more quantitative approaches by detecting general trends in the data and combining existing variables to form new ones according to a set of equation formation rules. Thus, if one trend is dominant, it will be detected even if it is not universal. Just how this is done shall be discussed in Chapter 3 where a theory of equation discovery is presented.

2.1.3. Competing Processes

Allowing the presence of multiple equations introduces a new problem, that of *competing processes.* The competing process problem occurs when there is conflicting data which prevents the discovery system from being able to discern the presence of any relations. Suppose, for example, that half of the events represent data for the relation x+y=0, while the other half represents x-y=0. As a result, x is decreasing as y increases for half of the data, while it is increasing with y for the other half. To a trend detecting algorithm, the events would effectively cancel each other out and force it to make the conclusion that x and y are independent of each other. The current work makes no direct attempt to solve this problem and therefore would be unable to resolve the above example. An indirect solution

[1] A three-dimensional approach would notice a 3-way relationship between x, y, and z and form xyz in a single step. Taking this to n-dimensions could prove to be impossible to implement, although the author knows of no such attempt.

does exist, however, as a consequence of the multiple equation assumption. In order to reason in the presence of multiple relations, ABACUS uses rather loose discovery criteria. Consequently, the presence of competing processes does not cripple the learning procedure if the number of observations of one process strongly dominates the other.

2.1.4. Guiding Search

The assumptions, or more appropriately the lack of assumptions made so far have now created some rather imposing problems concerning the search space that must be investigated. Combining existing variables to form new ones is inherently exponential. Allowing irrelevant variables introduces the possibility of many blind alleys, as large subtrees may be formed from an equation which contains an irrelevant variable. Allowing multiple equations to exist makes it difficult to know when the best possible relation has been found short of performing an exhaustive search and then choosing the best one. Also, the problem of deciding which equation is best becomes more complex.

There are a number of possible ways to solve these problems. As stated earlier, we might require the user to provide formula templates to help limit the search and to provide a way of measuring the quality of an equation. We might also require that the user indicate what variables are dependent and which are independent. This would help guide and limit the search by providing some knowledge of causality, but suffers from the same flaw as the relevant variables assumption. In a scientific discovery environment, little may be known about the attributes involved. We might provide the learning mechanism with background knowledge of the domain such as what types of relations are typical for this domain or knowledge about the physical meaning of an entity having particular units. While these approaches appear somewhat less restrictive than the formula template method, they still violate the goal of this work. To limit search, ABACUS uses a number of domain independent constraints which only prevent mathematically redundant or physically impossible equations from being formed. Likewise, the few heuristics used deal with general mathematical principles.

Allowing discovered equations to hold for only a subset of the observed events in such an unrestricted search environment still leaves the problem of knowing when to stop. Having discovered an

equation which describes a certain percentage of the events, how is the learning procedure to know there isn't a better solution further on? Accepting a "good" but imperfect answer leaves one with the problem of knowing when good is good enough. Indeed, how can the "goodness" of an equation be measured? We might look at the form of the equation itself and derive a measure of how syntactically appealing or unappealing it is. However, this might prove to be costly and arbitrary. Likewise, we might examine how many attributes or which attributes are involved in the equation and make a decision based on these factors. Alternatively, the percent of the observed events described by the equation could be calculated, evaluating a relation on how general it is for describing the observations. This last approach is the one taken here and is discussed further when search is examined in detail in Chapter 3.

2.1.5. Knowing When to Use a Discovery

One final issue remains. In Figure 2, two equations were found which applied to disjoint subsets of the data. Much information would be lost if the program merely reported that the two equations described the given data. In addition to producing the two equations, the program should also indicate which subset of the data each equation corresponds to. Such a specification would provide a constraint on the applicability of discovered equations. Using the graph of Figure 2, one could say that one equation describes the points on the curve while the other describes the line. This description is not precise enough, however, because it does not refer to the values of the points themselves. In addition, a program could not describe the values for x and y in this way because it cannot "see" the graph. Suppose a new observation was made. Without reconstructing the picture, one could not know which equation applies to it short of testing each equation in turn.

IF x = 0..5
    THEN y = x^2
IF x = 5..30
    THEN y = 30 - x

Figure 3. Domain Constraints for Graph Example

In response to this problem, when multiple equations are discovered, ABACUS associates a logical condition with each equation in terms of the observed attributes. These conditions are called *domain constraints.* For example, Figure 3 shows domain constraints which apply to the graph of Figure 2. If a point is observed whose x value is less than 5, then $y=x^2$. If the x value is between 5 and 30, then $y=30-x$.

2.2. Related Work

Little work on quantitative empirical learning exists and consequently only moderate progress has been made. The most recognized work is that of Pat Langley and his colleagues in constructing a series of programs named BACON [Langley, 1981a, 1985]. This work began in 1978 with the construction of BACON.1 and continues today with the BACON.6 system. We shall examine each version in turn.

The production systems BACON.1 and BACON.2 were limited in the types of discoveries they could make [Langley, 1981a]. This was due primarily to the distinction made between observed data and hypotheses proposed to describe the data. Specifically, hypotheses could not be used to generate other, more general hypotheses.

BACON.3 [Langley, 1979, 1981a] was able to formulate equations representing a number of laws of Physics and Chemistry. It introduced the concept of treating hypotheses generated at one level as variables for hypotheses at higher levels by allowing *levels of description.* This enabled it to discover more complex laws. The ideal gas law (PV/NT=8.32), for example, must be built up in a layered fashion by creating the hypothesis PV, using this and the directly observed attribute T to form a more general hypothesis PV/T and so on until PV/NT is formed which summarizes all of the given data. BACON.3 built hypotheses by detecting when a variable's values were directly dependent upon those of another variable. If this relationship could be described in terms of a line with a constant slope, a hypothesis for the slope term ((y-i)/x) would be created as well as a term for the intercept (y-mx) if non-zero. In the absence of such a line, a simple product or quotient term would be created.

BACON.3 consisted of 86 productions divided into 7 major categories (Figure 4). As will be shown later, these categories are roughly similar to those used in the ABACUS system. BACON.3 required

1. A set for gathering directly observable data;
2. A set to detect regularities in the data generated by the first and fourth sets;
3. A set that calculates the values of theoretical terms;
4. A set that checks for loops by comparing new theoretical terms to existing ones;
5. A set for noting related theoretical terms and ignoring their differences;
6. A set to collapse clusters with identical conditions;
7. A set for discovering irrelevant variables and ignoring their values.

Figure 4. Production Rule Categories of BACON.3

enough data to be gathered so it could always be able to observe changing values of two variables while holding the others constant. The user was also asked to identify which variables should be treated as dependent and which as independent. The approach taken in ABACUS has been to not require an exhaustive list of all permutations of variable values, although certainly some must be provided to detect trends in the data. For simple or contrived experiments, it is not too much to ask that the user provide all permutations of variable values. For original and sometimes complicated or expensive experimentation, this may be an unreasonable requirement.

While irrelevant events were not allowed, BACON.3 made an effort to deal with irrelevant variables by including productions which noticed when a dependent variable remained constant while a particular independent variable was varied. If such an occurrence was discovered, the independent variable was deemed irrelevant and effectively ignored in the future. While this scheme addresses the issue of irrelevancy, it only addresses those irrelevant variables which never have any effect upon the other variables' values. It does not address the problem of the presence of variables which are actually related to some other variables, yet are not part of the desired relation. For example, the mass of a ball is related to its radius, yet may not be required information for the final solution.

BACON.4 [Langley, 1983a] built upon the methodology of BACON.3 by adding two new concepts. First, the ability to postulate intrinsic properties for symbolic entities was introduced to enable it to reason in situations where a nominal (symbolic) independent variable influenced the value of a numeric dependent variable. This gave BACON.4 the power to discover a version of Ohm's law (I=V/R) when

given the current I as the only numeric variable and two nominal variables representing different batteries and different wires. Secondly, BACON.4 had the ability to detect when common divisors existed for a variable's values. This enabled it to discover the relative atomic weights of Hydrogen, Nitrogen, and Oxygen. However, BACON.4 required noiseless data which greatly simplifies these reasoning tasks.

BACON.5 [Langley, 1981b] included a simple mechanism for learning by analogy. This simplified the learning of symmetric laws such as conservation of momentum. For a two object collision, conservation of momentum may be stated as

$$m_1(v_{1i} - v_{1f}) = m_2(v_{2f} - v_{2i})$$

Once the relation $m_2(v_{2f}-v_{2i})$ is found, $m_1(v_{1f}-v_{1i})$ would be postulated via analogical reasoning. BACON.5 also made steps to allow a significant amount of noise in the data presented, but at the cost of eliminating the possibility of irrelevant variables.

The most recent of the series, BACON.6 [Langley, 1983b, 1985], deviated somewhat from the methodology of the previous systems. First, the form of the law must be provided by the user. This restricts the search space and allows the system to consider more complex laws, such as $\sin(y)=ax+b$, if the user so desired. Search moves up a pregenerated permutation tree for the observed variables' values, combining terms according to the formula forms provided by the user. BACON.6 differs from its predecessors in that it can discover more complex formulas and has the ability to include trigonometric and algebraic functions. As with BACON.5, BACON.6 allowed no irrelevant variables in the data provided.

Of possible future significance to quantitative learning is the field of Dimensional Analysis. Dimensional Analysis [Langhaar, 1951; Huntley, 1952] is an established discipline used primarily in the fields of fluids and heat transfer theory. It can be used to generate formulas that describe a given physical phenomenon when a sufficient amount of prior domain knowledge is already known. The procedure first requires that the set of relevant variables is known. The complete set of dimensionless equations possible from these is then generated. From the set of dimensionless equations the formation

of the appropriate equation is often a straightforward task. Because of the amount of domain knowledge required and other possible limitations unknown to the author at this time, the usefulness of dimensional analysis for quantitative empirical learning is unclear.

Mieczyslaw Kokar [Kokar, 1981] uses Dimensional Analysis in his work on quantitative learning. Kokar's method is intended to determine "the completeness of the set of parameters characterizing the given physical process." For this purpose, he introduces the concepts of *similarity measure* and *weak essentiality*. The similarity measure allows him to state that the current set of parameters is insufficient to characterize the observed process. Conversely, weak essentiality provides a measurement on whether a particular attribute should be included in the required set of parameters.

2.3. System Overview

A block diagram of the ABACUS system is shown in Figure 5. ABACUS consists of three semi-independent modules. The *user interface* consists of a menu from which a variety of tasks may be selected. Under normal usage, the system is presented with empirical data and the *equation learning module* is invoked to discover equations for disjoint subsets of the events. Upon completion of this phase, the *domain constraint module* is presented with the resulting class sets so that it may generate descriptions for each class which enable one to know when to apply each of the discovered equations. The result is a series of if-then rules which state that if for any event the set of boolean expressions for the attributes in the description is satisfied, then this equation describes the event. In addition, the equation discovery module may be invoked separately by the user if one simply wants to see what equations result from different parameter values. Similarly, the domain constraint routine may be called repeatedly with different parameter settings once the data has been divided into classes by the equation discovery module.

The quantitative learning module searches for the best equation to describe the given empirical data. If the discovered equation holds for all events, the learning task is completed and no domain constraints can be generated. However, if only a subset of the events are described, then the events not in this set are still not described by an equation. The operation of the equation learning module may

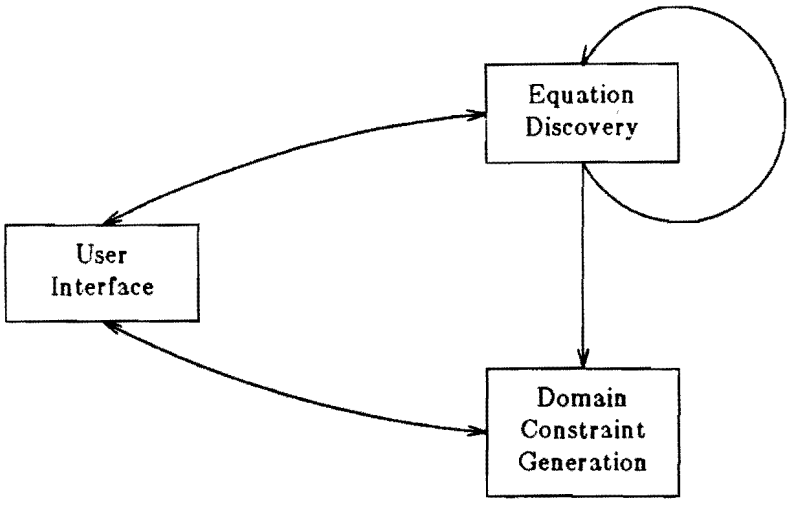
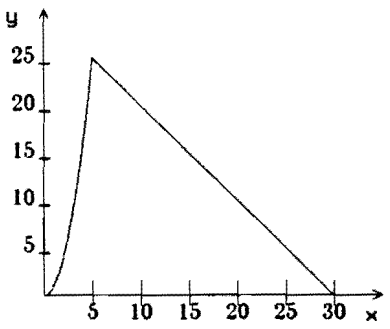


Figure 5. ABACUS System Overview

then be described as follows. Given a set of data, the module will look for the best equation to describe the events. The subset described by this equation is then removed from the list of events and placed in its own unique *class set* along with the equation describing it. Often, the final equation may describe several groups of events where the value of the equation differs in each group. When this occurs, a number of class sets are formed, one for each constant value. Any remaining events, namely those not inserted into a class set, are then passed to the equation learning procedure again in the hope that a different relation may exist for these events. This iterative process repeats until no more events remain or until no more relations can be found. In the latter case, the remaining events are placed in a "miscellaneous" class set. The end result is a classification or clustering of the given observations according to the relations they represent.

Once the data has been divided into classes, the domain constraint algorithm is used to generate discriminant descriptions for these classes. This enables the system to characterize the equations semi-independently of the observations which were used to generate them and thus adds predictive power to the program's conclusions. A detailed example will clarify how all the modules operate and interact.



(a)

ClassA	Cover:	[x = 0.10 .. 5.00]
	Relation:	$x^2 = y$
ClassB	Cover:	[x = 5.10 .. 30.00]
	Relation:	x + y = 30

(b)

Figure 6. ABACUS Analysis of Graph Example

Let us examine the process by which ABACUS would analyze the graph example reproduced in Figure 6(a). First, the observed values for x and y are read in and the equation discovery module is invoked. As there are only two variables, search is quite simple. First, the values for x are found to roughly increase when the values for y increase,[2] causing the creation of the formula x/y. Further search discovers that x decreases as x/y increases and so $\frac{x^2}{y}$ is generated. While this relation takes on a constant value for a large percentage of the data (70%), search continues in the hope that a more general relation exists. Further search, however, fails to uncover a relation which holds for more than 70% of the data and so $\frac{x^2}{y}$ is returned and a class set is created for the appropriate events. The equation module is again invoked and search proceeds in a similar manner on the remaining events. This time, y decreases when x increases and x+y is created. As x+y equals 30 for all events, the relation is returned immediately and another class set is created. Because there are no remaining events, equation discovery

[2] As stated earlier, ABACUS does not solve the competing processes problem. One may observe in the figure that the values for y go up with the values of x for the curve while they go down as the values of x go up for the line. In this example, there were 16 points given for the curve and 7 given for the line.

is completed and the domain constraint module is called to discover what general properties of the data can be used to distinguish between the two class sets. The results can be seen in Figure 6(b). They state that when x is below 5, $y=x^2$, but when x is between 5 and 30, $x+y=30$.

3. EQUATION DISCOVERY

The traditional search paradigm is presented as a process of repeated expansions of individual leaves in a tree. Each expansion adds to or modifies the leaf in a single step which is independent of the existence of other (nonancestor) nodes in the tree. As a result, the issues to face when deciding on a search strategy traditionally involve the probable breadth of the tree, how deep the goal is likely to be, and the availability of a reliable evaluation function or heuristic. Quantitative empirical learning by contrast possesses a unique search paradigm. It is this uniqueness which makes search probably the single most troublesome aspect of quantitative discovery. A graphical representation of a typical search space for quantitative learning is depicted in Figure 7.

The uniqueness of quantitative empirical learning lies in the fact that expansion of leaf nodes is not independent of the existence of other nodes in the tree. First, the expansion process involves taking a node and comparing it to other nodes in the tree. All nodes in this domain require two parents rather than the normal single parent. As a result, a node cannot simply be expanded irrespective of order, for its expansion may depend upon the existence of another node. This is a very important point which rules out most forms of depth-first search and complicates other popular forms of search such as best-first. Secondly, given that N nodes currently exist, one node could conceivably be responsible for over N-1 new children nodes. Thus each level in the tree should be expected to be perhaps an order of magnitude larger than the previous level.[3] This leaves obvious problems for a simple breadth-first approach. Improvements for breadth-first search, such as using a beam technique, are hampered by the lack of reliable evaluation functions and thus the fact that any given node may lie on the path to the final solution. As this path may prove to be the only one available, removal of nodes may prevent the program from finding any solution or perhaps prevent it from finding the best solution.

[3] We will use the familiar term of search tree even though the two-parent situation means that the object of discussion is more properly called a graph.

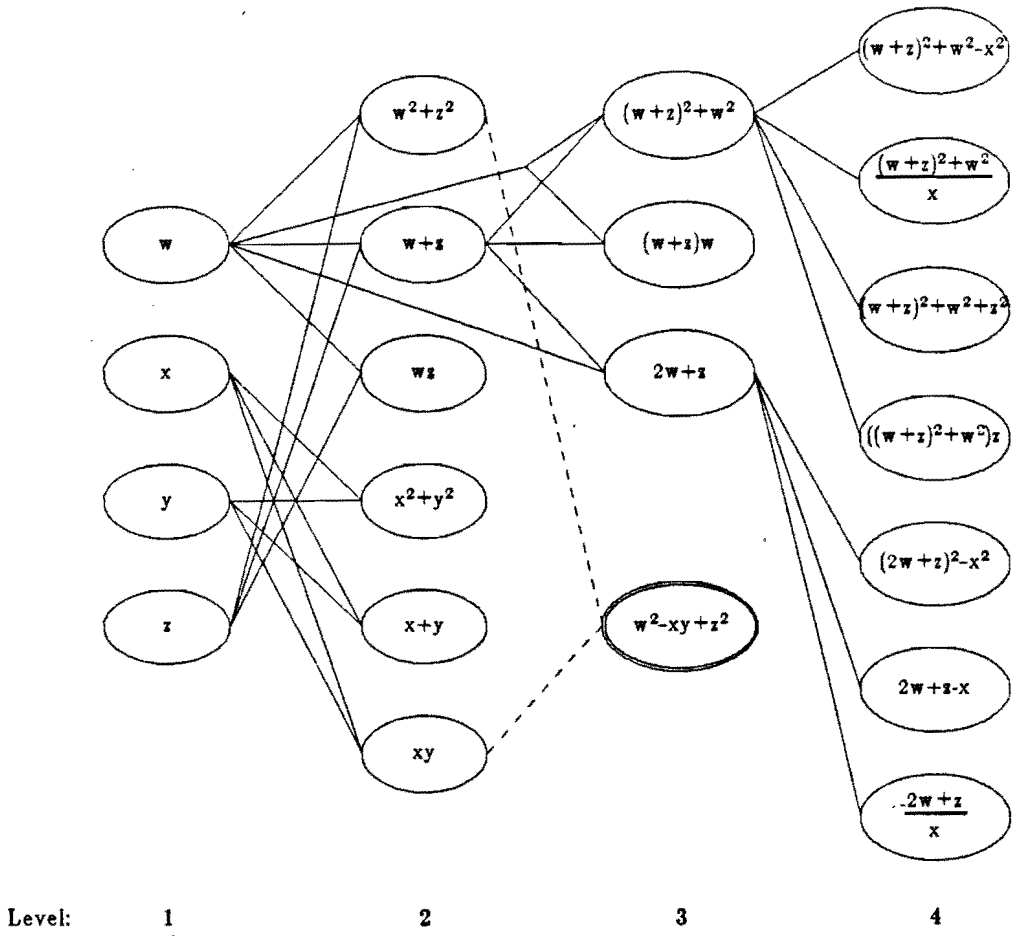


Figure 7. Partial Search Space for w^2-xy+z^2

3.1. Variable Relationships and Qualitative Proportionalities

At the heart of quantitative discovery is the concept that one variable's values may be dependent in some way upon the values of another variable. When the effect two variables, x and y, have on each other can be isolated by holding all other conditions constant, these variables may be said to be *qualitatively proportional* if the values of x rise when the values of y rise. Similarly, x and y are *inversely qualitatively proportional* if x decreases as y rises.[4] With this definition, there are four assertions possible

[4] Notice that no dependency ordering is implied here and thus no causal relation in the strict sense. [Forbus, 1984] defines qualitative proportionalities in such a way that causality is taken into account. Under his scheme, x would be qualitatively pro-

as the result of a qualitative proportionality measurement:

```
prop+(x,y)   - x and y are qualitatively proportional
prop-(x,y)   - x and y are inversely qualitatively proportional
noprop(x,y)  - x and y are unrelated
prop?(x,y)   - insufficient data to determine if x and y are related
```

3.1.1. Proportionality Graph

From these assertions we may construct an undirected graph, called a *proportionality graph*, where the nodes represent variables and edges indicate the presence of a qualitative proportionality relation between their incident vertices (Figure 8). For our purposes, edges shall only be constructed for prop+ and prop- relationships and prop? will effectively be treated as noprop. In Figure 8, a is proportional (+ or −) to b, but not proportional to c.

The use of proportionality graphs is examined in section 3.5. At the present, let us define what it means to have cycles in the graph. In this context, the term cycle refers to any biconnected components [Aho, 1974] which may exist in the graph. A biconnected component refers only to the maximal cycles in the graph, or in other words, only those cycles which are not a subset of some other cycle. In Figure 8,

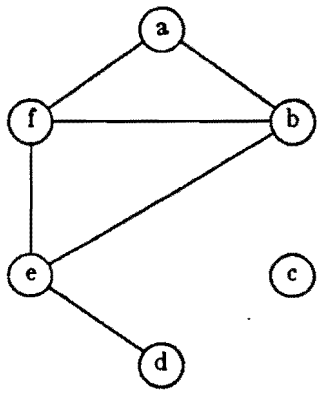


Figure 8. Proportionality Graph

portional to y, but y would not be qualitatively proportional to x.

the single cycle (or biconnected component) consists of the set of nodes {a b e f}. Under the traditional definition of a cycle, {b e f} and {a b f} would also be considered cycles, but for our purposes they are redundant with respect to the larger superset.

3.1.2. Establishing Qualitative Proportionalities

To make a qualitative proportionality assertion about variables x and y, ABACUS looks for general trends in the data. The trend detection algorithm sees if y rises or decreases as x rises when all other variables are held constant. Holding the other attributes constant removes any dependencies x and y may have on these variables, thus isolating the effect x and y may have on each other. However, if variable y (or x) is a program generated variable composed of user defined variables a and b, then a and b are removed from the set of variables which must be held constant. Since they are necessarily dependent upon y, it would be impossible to hold a and b constant while changing y. An *exclusion set* is then defined to be the set of attributes which do not need to be held constant. To measure the relationship between variables x and y, the events are sorted using a multi-key sort. The keys are defined to be

$$\text{Keys} = \{\text{User Defined Attributes}\} - \textit{exclusionset} - \{y\}$$

x is the last key used in the sort so that for groups of events where all attributes other than x, y and the exclusion set are constant, x is in increasing order (see Figure 9). If no groups can be found where all of the other variables do not change value, then prop? must be asserted. For each of the groups found, a measurement is made of the monotonic relationship between x and y by counting the number of times y increases and the number of times y decreases. From these counts, a relationship measurement is calculated which ranges from 0 to 100. A value of 0 indicates the highest level of indirect proportionality while 100 indicates the highest level of direct proportionality. Once all groups have been so measured, the average is obtained and used as the measure of relevance between x and y. If the value is low enough, prop- is asserted and if the value is high enough, prop+ is asserted (see Figure 9). Otherwise, a conclusion of noprop is made. As can be seen, the proportionality criterion is rather loose, allowing a moderate degree of noise and a limited amount of competing processes.

v	w	x	y
5.2	10.7	100.0	437.0
6.7	8.9	138.5	420.1
6.7	8.9	145.2	449.8
6.7	9.4	149.7	446.6
6.9	9.2	153.0	445.2
7.1	8.3	162.3	467.2
7.1	8.3	163.9	470.5
7.8	8.0	160.7	479.6

Figure 9. Data Groupings Showing prop+(x,y)

3.2. Equation Formation - A Search for Constancy

The goal of asserting qualitative proportionalities between variables is to establish what possible mathematical relationships exists between them. If we knew that the value of x always went down when the value of y went up, then xy = constant might be the relation we are looking for. This may be generalized to say that if prop-(x,y), then xy forms a variable which is more likely to take on a single value than x or y did.

With this concept in mind, search in quantitative discovery involves the continual combination of variables which are qualitatively proportional to form new variables in the hope of finding a variable which takes on a constant value. There are several variable creation rules which may then be used to guide such a search. At the highest level are the rules which deal with qualitative proportionality assertions:

If prop+(x,y) then
Generate a difference relation between x and y (1)
Generate a quotient relation between x and y

If prop-(x,y) then
Generate a product relation between x and y (2)
Generate a sum relation between x and y

These rules expand upon the variable combining concept presented above. Notice that they will always

direct search to create variables having higher degrees of constancy. These rules are similar to those used in the BACON system [Langley, 1981a, 1983a] and also parallel the use of the M-descriptor defined in [Michalski, 1983].

What is needed now are rules which tell how to create the relations that rules (1) and (2) say to generate:

To Generate Product(x,y) => (3)
    Create x*y

To Generate Quotient(x,y) => (4)
    Create x/y

To Generate Sum(x,y) =>
    Create x+y (5)
    Create x^2+y^2
    Create x^n+y^m where $n \neq m$

To Generate Difference(x,y) =>
    Create x-y
    Create x^2-y^2 (6)
    Create x^n-y^m where $n \neq m$
    Create y-x
    Create y^2-x^2
    Create y^m-x^n where $n \neq m$

While rules (3) and (4) are obvious, (5) and (6) are somewhat more complicated. The powers are included to compensate for the fact that a variable cannot be qualitatively proportional to itself. Were the final formula to be x^2/y, then x/y could be inversely proportional to x to create the x^2 term. With x buried inside of an additive relation (e.g. $x+y$), however, an x^2 term could never be created (e.g. x^2+y^2). Higher order pairs such as x^3+y^3 cannot be generated. This is simply a limitation of the program, albeit a reasonable one since these higher powers rarely appear in the natural sciences. The redundancy of (6) accounts for the fact that negating a variable negates its proportionality to another variable. If $(y-x)/z$ was the desired relation and the program created only $x-y$, $(x-y)z$ would be formed rather than the appropriate quotient relation. It is hoped that a better solution can be found for both of the above problems. While it appears that (5) and (6) generate an excess of variables, never will all of their actions be taken. Constraints controlling the use of the above rules are discussed in the next section.

With these six rules, search now has the ability to generate new variables when given the qualitative relationships which exist among the current variables. The details of how these rules are applied to qualitative proportionality assertions are presented in section 3.5.

3.3. Constraints

Several domain independent constraints are used to control the large search space involved with quantitative learning. These constraints involve eliminating mathematically redundant expressions and physically impossible relationships. The constraints are divided into three categories as follows:

- Units Compatibility Rules
- Redundancy Detection
- Tautology Detection

3.3.1. Units Compatibility Rule

When the system decides to create new variables by firing the rules presented in the previous section, the additive relation rules (5 and 6) will attempt to create a variety of new variables. Were all of these variables to be created every time one of the rules fired, the number of variables would grow so quickly that the search space would soon become unmanageably large. A simple physical constraint drastically limits the possible actions. For two entities to be added or subtracted they must be the same type of entity, that is, they must be in the same units. One may divide meters by seconds, but not subtract seconds from meters. Any action which violates this *units compatibility rule* will be blocked. This is similar in intent to the dimensional cohesiveness requirement of dimensional analysis [Langhaar, 1951; Huntley, 1952].

Whenever a variable is created, its units are calculated and stored with it. This allows the system to quickly compare the units of two items. Examining the sum relation rule (5), if the units of x and y are equal, then the first two actions, namely $x+y$ and x^2+y^2 , are allowed to occur. The third action, x^n+y^m will not be allowed to take place as there are no two distinct values for n and m in which the units for x^n and y^m would be compatible. If, on the other hand, the units of x and y are not the same, but differ only

in magnitude such that the units of x^n is compatible with the units of y^m, then the third action would fire and the first two would be blocked.[5] Finally, if the units are not equal and can not be coerced into being compatible, then none of rule 5's actions are allowed, effectively blocking the entire rule. In practice, rules 5 and 6 are usually blocked. It should be pointed out that these constraints only test the identity of units and provide no semantic interpretation to guide the search.

3.3.2. Formula Redundancy

A common side effect of combining existing variables to form new ones is the possibility that for any new variable, a mathematically equivalent yet syntactically different variable may have already been created. This is especially likely since variables created at one level in the search may be combined with existing variables from any other level. For example, say variable x represents the relation:

$$\frac{(ab)}{(cd)}$$

where the parentheses show that x was created by dividing a variable ab by another cd. Further, suppose during the course of the search the variable b/(cd) had been created. At some point, the system will then try to create a new variable y:

$$a\left(\frac{b}{(cd)}\right)$$

As we can see, x and y represent the same variable and if the system creates y, it would be wasting time and complicating the search space. From a purely syntactic examination, however, x and y are not equal. How then can the program know if an existing variable is equivalent to the one it is trying to create?

What is needed is a canonical form for expressions so that equivalent formulas will always be syntactically equal. Then formulas such as those for the variable x above may be stored in an assertional database which can be examined for redundancy each time a new variable is to be created. The canonical form of an expression is given by the grammar of Figure 10, where a variable's formula

[5] A further constraint requires that both n and m be less than 4. Again, this is a heuristic limitation, but seems reasonable given that higher powers are rare in the natural sciences.

```
Form          ::=  Typed_Form | Generic_Form
Generic_Form  ::=  ( NIL ( Identifier ) NIL )
Typed_Form    ::=  ( Form_Type Form_Side Form_Side )
Form_Type     ::=  "/" | "-"
Form_Side     ::=  ( Form_List ) | NIL
Form_List     ::=  Identifier Form_List | Identifier
Identifier    ::=  variable name | form name
```

Figure 10. *Form* Context Free Grammer

represented by this grammar is called its *form*. The most common version of a form is the *typed form*. A typed form represents either a multiplicative relation or an additive relation. In forms of type "/", all variables in each *form side* (referred to as `left(form)` and `right(form)`) are multiplied together. The form then represents the product of all terms on the left side divided by the product of the right side terms. Thus:

$$(/\ (a\ b)\ (c\ d)) \quad <=> \quad \frac{ab}{cd}$$

Forms of type "-" are defined similarly:

$$(-\ (u\ x)\ (y\ z)) \quad <=> \quad (u + x) - (y + z)$$

Notice that the "variables" in a form may represent either user defined variables or the name of a form whose type is **opposite** that of the form it is in. When a variable having a multiplicative form is involved in an additive relation, its form must be replaced by the atom name associated with it. Thus u in the above form might be the name of a multiplicative form.[6]

Generic forms only represent a single item and thus do not require a type. These forms are never asserted in the database as the lack of type indicates that they do not represent a variable generated by the program. It should also be pointed out that the lack of type means that they are compatible with both of the typed forms and so do not have to be named when combined with one. A generic form is

[6] When each form is added to the database, its unique name is asserted with it and the form is likewise added to the name's property list. Because we are guaranteed no two identical forms will exist, we are guaranteed that each unique name will in fact refer to a unique form.

created for each user defined variable.

In addition to adhering to the above syntax, forms must also always adhere to a sum-of-products format. For example, if $form_1$, (- (a) (b)), was multiplied by $form_2$, (/ (x y) nil), the correct new form would be (- ($form_3$) ($form_4$)) where $form_3$ is the name for (/ (a x y) nil) and $form_4$ is the name for (/ (b x y) nil). While (/ ($form_1$ x y) nil) is an equivalent form, it is not in sum-of-products format.

3.3.3. Numerical Tautologies

Another problem with combining mathematical formulas is the possibility that a mathematical cancellation may result, causing the program to effectively take a step backwards. Say for example the

---

Operation = "*"

(1) type(form1) ∈ { /, nil } ∧ type(form2) ∈ { /, nil }
  ∧ (left(form1) ∩ right(form2) ∨ right(form1) ∩ left(form2))

(2) (type(form1) = "-" ∨ type(form2) = "-") ∧ type(form1) ≠ type(form2)
  ∧ [(∀ item ∈ sum_form) (left(product_form) ∩ right(item))
    ∨ (∀ item ∈ sum_form) (left(item) ∩ right(product_form)
      ∨ item ∈ right(product_form))]

Operation = "/"

(1) type(form1) ∈ { /, nil } ∧ type(form2) ∈ { /, nil }
  ∧ (left(form1) ∩ left(form2) ∨ right(form1) ∩ right(form2))

(2) (type(form1) = "-" ∨ type(form2) = "-") ∧ type(form1) ≠ type(form2)
  ∧ [(∀ item ∈ sum_form) (right(product_form) ∩ right(item))
    ∨ (∀ item ∈ sum_form) (left(item) ∩ left(product_form)
      ∨ item ∈ left(product_form))]

Operation = "+"

(1) type(form1) ∈ { -, nil } ∧ type(form2) ∈ { -, nil }
  ∧ (left(form1) ∩ right(form2) ∨ right(form1) ∩ left(form2))

(2) (type(form1) = "/" ∨ type(form2) = "/") ∧ type(form1) ≠ type(form2)
  ∧ product_form ∈ right(sum-form)

Operation = "-"

(1) type(form1) ∈ { -, nil } ∧ type(form2) ∈ { -, nil }
  ∧ (left(form1) ∩ left(form2) ∨ right(form1) ∩ right(form2))

(2) (type(form1) = "/" ∨ type(form2) = "/") ∧ type(form1) ≠ type(form2)
  ∧ product_form ∈ left(sum-form)

Figure 11. Conditions for Numerical Cancellation

---

program discovers prop-(a/b, bc). Creating a new variable $\frac{a}{b}$bc would result in b canceling out and work being wasted. In addition, were such operations allowed to go unchecked, the system may soon discover that $\frac{abc}{abc}$ always equals 1 for any data given. While this may be an interesting observation if we were trying to learn principles of mathematics, it defeats the purpose at hand.

ABACUS allows no action which would result in a mathematical cancellation. Using the syntax for formulas presented in the previous section, a check for tautologies is reducible to a set of simple logical conditions. These conditions are stated in Figure 11. If either condition for a given operation evaluates to true, the proposed action is blocked. In the figure, "Operation", "form1", and "form2" refer to the relation being created, (e.g. for x-y, form1 refers to x, form2 refers to y, and the Operation is subtraction). When the two forms have a different type (other than NIL), the one having type "/" is designated the product_form and the one having type "-" is called the sum_form. For this case, order is not of importance.

3.4. Recognizing the Goal

Because ABACUS allows irrelevant events, recognizing when a good equation has been found and when to terminate search is not as easy as it would be otherwise. Consequently, there are three types of goal nodes accepted by the system. The first type of goal node (i.e. equation) is easily recognized. If an equation that describes all events is found, a final answer must have been found because no other equation could possibly describe a larger portion of the events. An equation that describes all events is one which evaluates to the same value (within a percentage range of uncertainty modifiable by the user) for every event. When such an equation is discovered, search terminates immediately. The second type of goal node also causes immediate cessation of the search process. A *nominal subgroup* is defined to be a set of events which are equal on all nominal attributes. If an equation is found which evaluates to a single value for a nominal subgroup, search terminates on the assumption that an equation of significance has been found. For example, in Figure 12(a), x/y takes on the same value for the nominal subgroup defined by those events whose object attribute equals "box". The third type of goal node does

object	x	y	x/y
circle	2	2	1
circle	4	3	1.3
circle	6	4	1.5
box	2	1	2
box	4	2	2
box	5	2.5	2
box	6	3	2
triangle	3	2	1.5
triangle	5	3	1.6

100% Constancy in a Nominal Subgroup
(a)

object	x	y	x/y
circle	2	2	1
circle	4	2	2
circle	6	3	2
box	2	2	1
box	4	2	2
box	5	2.5	2
box	6	3	2
triangle	3	2	1.5
triangle	5	2.5	2

67% Constancy for Entire Event Space
(b)

Figure 12. Sample Goal Node Recognition

not halt the search algorithm. As each new variable is created, its degree of constancy (largest number of events having a single value) is measured and the variable having the largest degree of constancy is always known. In Figure 12(b), x/y has a 67 percent constancy because six out of the nine events are equal to two. If two variables have the same constancy value, only the one discovered first is remembered as it is probably of a simpler and thus more desirable form. If search proves to be exhaustive in terms of its allowed limit, the equation having the highest degree of constancy is returned if it describes a sufficient percentage of the events as specified by a user modifiable parameter. If its constancy is not high enough, then the search algorithm will indicate that no formula could be found.

3.5. Search

The next few sections describe several search algorithms which appear to be useful for quantitative learning and have been tested in the ABACUS program:

- Breadth-First Search
- Proportionality Graph Search
- Breadth-First Search with Proportionality Graph Search
- Suspension Search
- Suspension Search with Proportionality Graph Search

While breadth-first search may be familiar to the reader, the later four search techniques have been developed in this work and are designed specifically for the search paradigm of quantitative learning.

To unify the discussion, three examples have been chosen which represent three major equation types. For these examples it will be assumed that search terminates as soon as the desired relation is found. The first example consists of data for the ideal gas law:

$$\frac{PV}{nT} = 8.32$$

where P is the pressure of the gas, V is the volume, T is the temperature in Kelvins, and n is the number of moles. The ideal gas law equation represents those relations consisting solely of multiplication and division and whose variables are all of degree one.

The second example is taken from Coulomb's law:

$$\frac{Fr^2}{q_1 q_2} = 4\pi\epsilon$$

This law relates the force of attraction or repulsion between two charged particles which are separated by a distance r. Coulomb's law is characteristic of those equations which consist solely of multiplication and division that include variables raised to powers greater than one. The presence of powers for this kind of equation will be seen to be significant.

The final example is the non-vector form of the law of conservation of momentum:

$$m_1v_1 + m_2v_2 = m_1'v_1' + m_2'v_2'$$

This form of conservation of momentum states that the total momentum of two particles which collide while travelling along the same line will be the same as the sum of their momenta after they collide. To complicate the example, the masses m_1 and m_2 will be allowed to change after impact producting m_1' and m_2'. Mass is given in kilograms while velocity is in meters per second. This relation represents all other equation forms, namely variables raised to various powers related by a mixture of addition, subtraction, multiplication, and division.

3.5.1. Search Technique 1: Breadth-First

From previous discussion, it would appear that, while the search tree is expected to be extremely wide, a traditional breadth-first approach is still called for in this situation. The final goal is almost always only a few levels down and the lack of good evaluation functions appears to prevent further improvement. As a result, the following breadth-first search algorithm was the first to be installed in the ABACUS program.

Using Figure 7 as an example, let us define level 1 as the user defined variables. Level 2 is created by combining variables from level 1 according to the six equation formation rules. Level 3 is then created by comparing each variable in level 2 with each of its siblings and with each of the nodes at higher levels (lower in number), namely level 1. Subsequent levels are then created similarly. As can be seen, even the mere existence of a node is quite costly, even if no children are ever sprouted from it. All nodes in the tree must be compared to it using the proportionality assertion algorithm, as well as any nodes yet to be created. Recall that in this domain leaf nodes are not merely expanded. Each node must be compared with every other node in the tree, even as the tree grows. In addition, each node may spawn many children. Thus the number of nodes at each level rises at an enormous rate. As a result, search was limited to go no deeper than level 4 in the assumption that any worthwhile relation should be found by then. This maximum depth is of course available as a user modifiable parameter, but for the discussions here it will be assumed to always equal 4.

3.5.2. Search Technique 2: Proportionality Graph

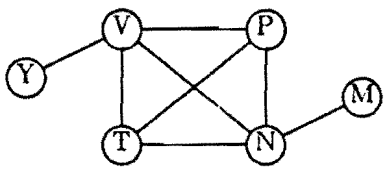
While breadth-first search appeared necessary, it suffers from some rather severe drawbacks. As no other traditional methods appear to offer a satisfactory solution, a new approach is needed, one which capitalizes upon the unique features of quantitative learning. Analysis of the kinds of equations being discovered shows that these equations may be divided into two categories. One type of equation is composed solely of multiplication and division, while all other types are composed of a mixture of additive and multiplicative relations. This later type of equation is rather troublesome since they are usually composed of relations from various levels (e.g. $m_1v_1+m_2v_2=m_1'v_1'+m_2'v_2'$) and tend to cause a lot of prop? assertions. The former equation class, however, possesses some rather useful characteristics. In the fields of physics and chemistry, the vast majority of physical laws lack addition or subtraction, so the first type of equation represents an important class of equations. Recalling the proportionality graph defined earlier, the equations falling in this category will form a cycle in the corresponding proportionality graph, baring the presence of an exorbitant number of prop? assertions. As an example, equation 1 below represents a general equation of this type.

$$\frac{abc^2}{def} = \text{Constant} \qquad (1)$$

Holding four of the variables in (1) constant and varying a fifth will necessarily cause the sixth to vary as well, in a direction which is completely predictable given the direction of change of the fifth and the respective positions of the fifth and sixth relevant to the major divisor. In the absence of prop? assertions, each variable is therefore qualitatively proportional (+ or –) to the other five. The subgraph for edges (a, b, c, d, e, f) in the possibly larger proportionality graph for the given problem must therefore be strongly connected and these nodes will thus form a cycle. This introduces another observation about the proportionality graph of such an equation. Irrelevant variables will be far more likely to be excluded from the above cycle, often being incident on perhaps only one edge. A search technique which has access to such a graph should direct its search to the interrelations of variables forming a cycle and avoid variables which are not contained in a cycle. Further, a multiplicative relation such as (1) may be formed in a traditional depth-first manner by multiplying in a single user

defined variable at each step (e.g. $ab \Rightarrow abc \Rightarrow \frac{abc}{d} \Rightarrow \cdots \Rightarrow \frac{abc^2}{def}$).

The second algorithm implemented in the ABACUS program is called a *proportionality graph search*. Once the initial graph is formed from the given user supplied variables, a simple graph traversal algorithm [Aho, 1974] is used to extract the cycles (biconnected components). The cycle sets are then sorted in decreasing order under the assumption that the largest cycles will prove to be the most promising. For each cycle set, a depth-first search with backtracking is performed where the depth of the search is given by the number of nodes in the set and for any such search path, each node in the cycle may appear only once. Powers of variables are therefore not possible after the first round of search. Backtracking is included to allow for the possibility of irrelevant variables in the cycle set. If no suitable relation can be found after examining each cycle set, a new proportionality graph is constructed whose nodes consist of all current variables, both those provided by the user and those generated by the program. Edges are formed again, where all edges from the previous graph are excluded from the new graph, cycles are extracted, and search proceeds as before. This process repeats either a maximum of four times or until a suitable relation is found.



(a)

Cycle Sets:
{ (P V) (P N) (P T) (V N) (V T) (N T) }
{ (Y V) }
{ (N M) }

(b)

Figure 13. Proportionality Graph for Ideal Gas Law

As an example of the heuristic power of this search technique, a sample proportionality graph is provided in Figure 13(a) for the ideal gas law, (PV/NT = 8.32), where a total of six attributes were provided by the user. As can be seen, the irrelevant variable mass, M, is independent of pressure, volume, and temperature, but is proportional to the number of moles of gas present. A similar situation exists for the variable Y. The 3 cycle sets of the graph are given in Figure 13(b), where solitary edges are simply treated as "cycles" having only one edge. The search tree resulting from the above strategy applied to this example is shown in Figure 14. For the ideal gas law, ABACUS happened to generate the minimum number of nodes possible to arrive at the correct solution. The breadth-first search strategy would have been able to discover the desired relation somewhere in level 3, but would have had to completely generate level 2, including those relations involving the two irrelevant variables.

While the proportionality graph search is quite adept at locating relations of this type, the ideal gas law example happens to be ideally suited to such a search technique. Other types of relations, even those composed solely of multiplication and division but with higher powers, are not so well suited to a

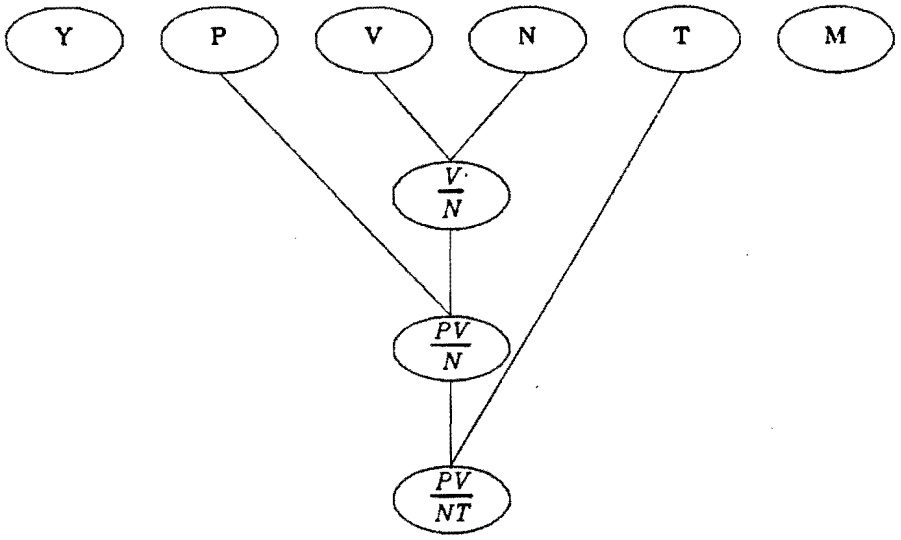


Figure 14. Proportionality Graph Search Tree for Ideal Gas Law

proportionality graph search. The difficulty begins with the second graph constructed and becomes increasingly worse with successive graphs. After completion of the first search pass, a large number of variables exist in the system, many of whose equations differ only slightly. Consequently, the second proportionality graph constructed has a much larger number of nodes than the first and, due to their similarities, the graph tends to be highly interconnected. Therefore, the extracted cycles are quite large, commonly encompassing the entire graph. As the depth of each cycle search is given by the cardinality of the set and backtracking must be allowed, a great deal of time is wasted exploring very deep levels of the search tree. However, the speedup achieved for examples whose relations may be found from the first round of search is greatly outweighed by the slowdown it causes for other examples.

3.5.3. Search Technique 3: Breadth-First with Proportionality Graph

For examples requiring only a single proportionality graph search pass, the proportionality graph search approach is indeed quite powerful. For more complicated examples, however, the approach suffers from a combinatorial explosion similar to that of depth first search. As progress from a node often depends upon what nodes have been created previously, a pure proportionality graph approach will waste a great deal of time on useless exploration. How can the benefits of this approach be maintained while discarding the negative aspects?

The next algorithm presented accomplishes this task by combining the previous two techniques. The new approach is virtually identical to the breadth-first algorithm described earlier, but with one important exception. Prior to breadth-first search, a single proportionality graph search pass is made on the initial data. All variables created during this pass are then considered as residing in level 1 along with the user defined variables and breadth-first search proceeds as before.

The benefits of such an approach may be seen in the following example (Figure 15). Given that the desired relation is Coulomb's law:

$$\frac{Fr^2}{q_1 q_2} = \text{Constant}$$

an unmodified breadth-first algorithm could not discover this relation until at least level 4. As each level

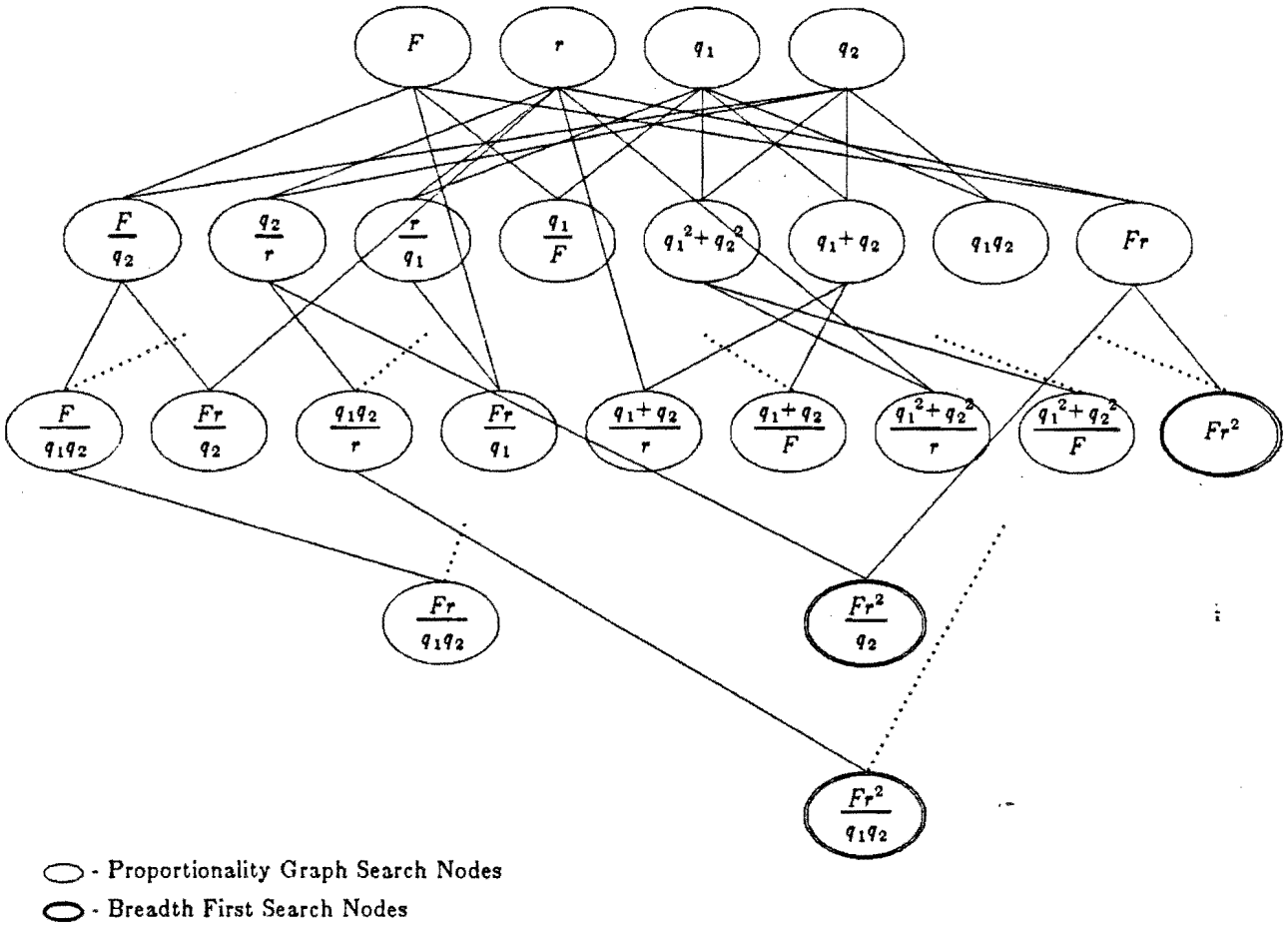


Figure 15. Search Tree for Technique 3 and Coulomb's Law

must be explored fully before proceeding on to the next level, this technique would generate many nodes and require a great deal of time. The pure proportionality graph approach would quickly generate $\frac{Fr}{q_1q_2}$ during the first pass, but because a universal relation cannot be found in the initial pass, many variables are generated in the process of looking for one. When the second pass begins, a very large cycle[7] must be searched in a depth-first manner, while all that is needed is one more r term which may not be found until the search permutations are nearly exhausted. A breadth-first examination starting from $\frac{Fr}{q_1q_2}$,

[7] In practice, the cycle often contains over 20 nodes.

however, will quickly encounter the desired relation. In Figure 15, the single ellipse nodes are those created by the initial proportionality graph search pass. A preorder traversal of these nodes will show the order in which the nodes were created. It can be seen that $\frac{Fr}{q_1 q_2}$ was the third node created. The breadth-first phase generated the three double ellipse nodes. The algorithm first visits the last node created, which explains why all breadth-first nodes were produced by the Fr node.

While it may appear that most of the problems are now solved, node existence is still very expensive. When a suitable relation cannot be found at all and the system must choose the best of what it has discovered as its final answer, search is exhaustive. While the depth of the breadth-first iterations is limited, search trees in this domain are still considerably wide and a great deal of time is spent exploring each level.

3.5.4. Search Technique 4: Suspension Search

Up to this point, the cost of a node's existence has been stressed, but no effort has been made to limit or modify node existence. In the introduction it was noted that no reliable evaluation functions exist and eliminating nodes is dangerous. A search technique would be of great value if it could remove nodes yet allow their return should they be needed. In response to these needs, a new search technique, called a *suspension search*, has been designed and installed in the ABACUS program.

Suspension search combines the benefits of a beam approach with the allowances for faulty heuristics provided by backtracking. Suspension search begins as a normal breadth-first search. At each level, however, the values for each node are examined. As the ultimate goal is to find a variable whose values are all or almost all the same, nodes possessing some degree of constancy would more likely lie on a terminating path than those nodes which lacked any degree on constancy. To this end, when each level is created, all nodes on that level are divided into an *active-nodes* set and a *suspended nodes* set. Search then proceeds on to the next level, where only the active nodes of previous levels are visible to the search algorithm. The next level is created by comparing all new active nodes of the current level with themselves and all active nodes of earlier levels (as well as with old active nodes of that level, as will

```
FUNCTION Suspension (ancestor_nodes, a_nodes, s_nodes, env) : boolean;
VAR
    new_a_nodes, new_s_nodes : node_list;

BEGIN
  if (env.level ≥ *Search_Limit*)              Search depth limit has been reached
    then return ( *Best_Constancy* ≥ *Constfactor* )

  if new_a_nodes and new_s_nodes can be created using a_nodes    Proceed with a_nodes only
    and the best constancy from these = 100%
        or Suspension (append(a_nodes, env.a_nodes, ancestor_nodes),
                       new_a_nodes, new_s_nodes, Descend(env)) )
    then return (true);

  if (level ≥ *Filter_Depth*)              Only a_nodes used after filter_level
     then save the environment
          return (false);
      end;

  if new_a_nodes and new_s_nodes can be created using s_nodes      Activate s_nodes
    then  if the best constancy from these = 100%
             then return (true);
          save the environment
          return ( Suspension (append(all nodes for this level, ancestor_nodes),
                               new_a_nodes, new_s_nodes, Descend(env)) );
     end;

  save the environment              Nothing found using a_nodes and s_nodes
  return (false);

END;
```

Figure 16. *Suspension Search* Algorithm

be seen shortly). If no relation has been found by the time the depth limit is reached, the best relation found so far is returned as the final result if its level of constancy is high enough according to a user supplied parameter. If not, search backtracks to the previous level where those nodes previously suspended are now activated and related to each other, the original active nodes, and those active nodes of earlier levels. Search then returns to the next level with a new set of active nodes. If still no relation is found, backtracking will go back farther and the process will repeat as before. An environment of each level is maintained to enable the program to remember what nodes were previously active when search returns. The suspension search algorithm is shown in Figure 16. When invoked initially, a_nodes is a list of the numeric user defined variables.

Now that fewer nodes are involved in the search at any one time and nodes may be characterized as either promising or unpromising, search may be allowed to explore deeper than before. Drawing from the heuristic continuation concept of game searches [Winston, 1984], a second search depth limit is defined, called the *filter depth*, which cites a limit shallower than that of the absolute depth limit. Search may proceed beyond the filter limit depth, but only active nodes are allowed for levels beyond this limit. Suspended nodes created at these levels are permanently discarded.

A sample suspension search tree is given in Figure 17 for the Coulomb's law example. The dashed horizontal line represents the filter limit depth. As can be seen, a number of nodes may be eliminated as a result of this technique and thus the search cost is considerably reduced.

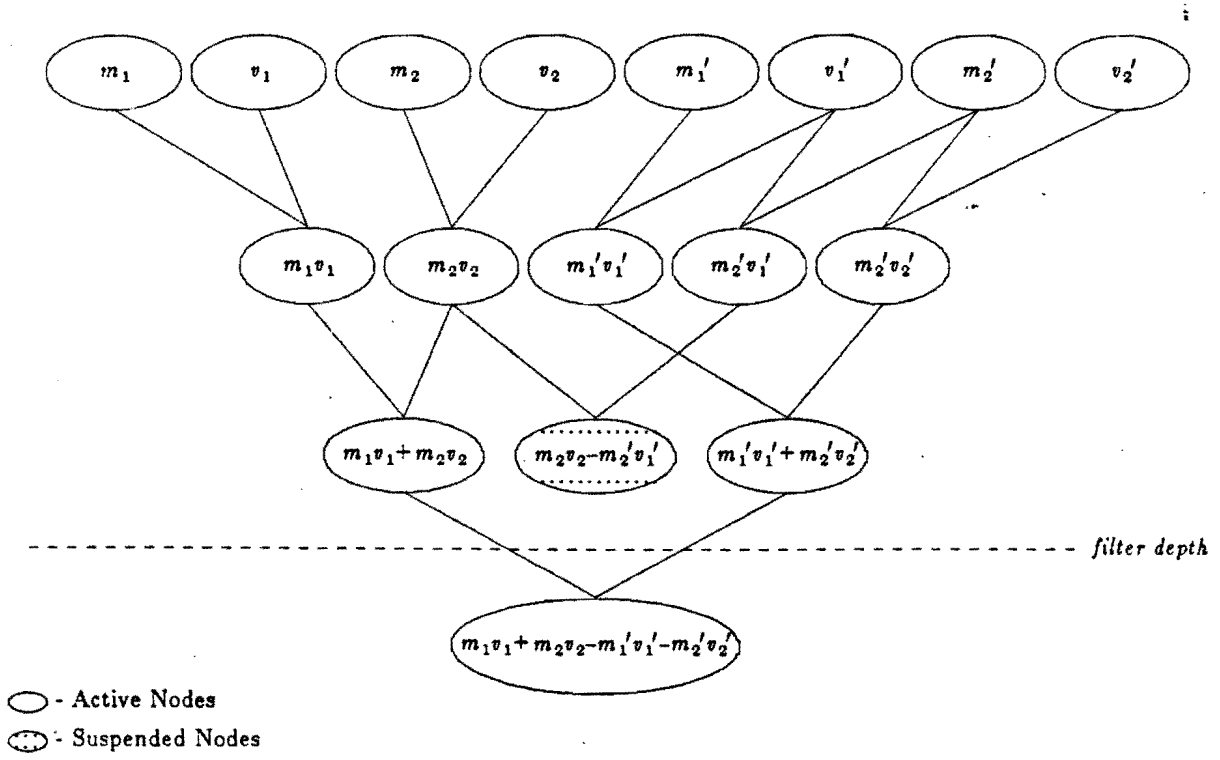


Figure 17. Partial Suspension Search Tree for Conservation of Momentum

3.5.5. Search Technique 5: Suspension Search with Proportionality Graph

While suspension search solves most of the major search problems, it is still basically a breadth-first search and lacks the heuristic power of the proportionality graph search. One final improvement on the ABACUS search algorithm modifies the straight suspension search by adding a single proportionality graph search pass, performed prior to beginning the main search routine. All nodes created during this phase join the user defined variables in level 1. If any of these nodes fail the constancy measurement, they will be designated as level 1 suspended nodes and processed accordingly.

This new technique favors quick discovery of laws which are composed solely of multiplication and division while still being able to discover more complicated equations in the same order of time as the pure suspension search algorithm. As cycles in the first pass can never be larger than the number of given attributes, the depth-first search of the first phase is not deep for most problems, thus creating variables which would normally be created for more complicated examples anyway.

3.5.6. Analysis

While the relative merits and failings of five search algorithms have been discussed, we shall now examine how they compare against each other for the three representative examples mentioned earlier. A rough ordering of the three examples has been established, based on the judged complexity of the problem. A graphical comparison of the five algorithms is presented in Figure 18, relating the complexity of each example to the number of nodes generated. Because the different search strategies create different search trees, the data does not provide a direct correlation. Work has been done, however, to have the breadth-first and suspension algorithms visit nodes in roughly the same order. For the two forms of suspension search, the number of nodes generated includes suspended nodes and therefore the full affect of the algorithm cannot be seen when measuring the number of nodes created. For complex examples, the suspension process speedup will be reflected by a decrease in the number of nodes generated at deeper levels, but for simple cases no such decrease would be seen.

For the ideal gas law example, the three proportionality graph methods went directly to the desired relation, discovering it in the minimum length of time. Breadth-first and pure suspension search

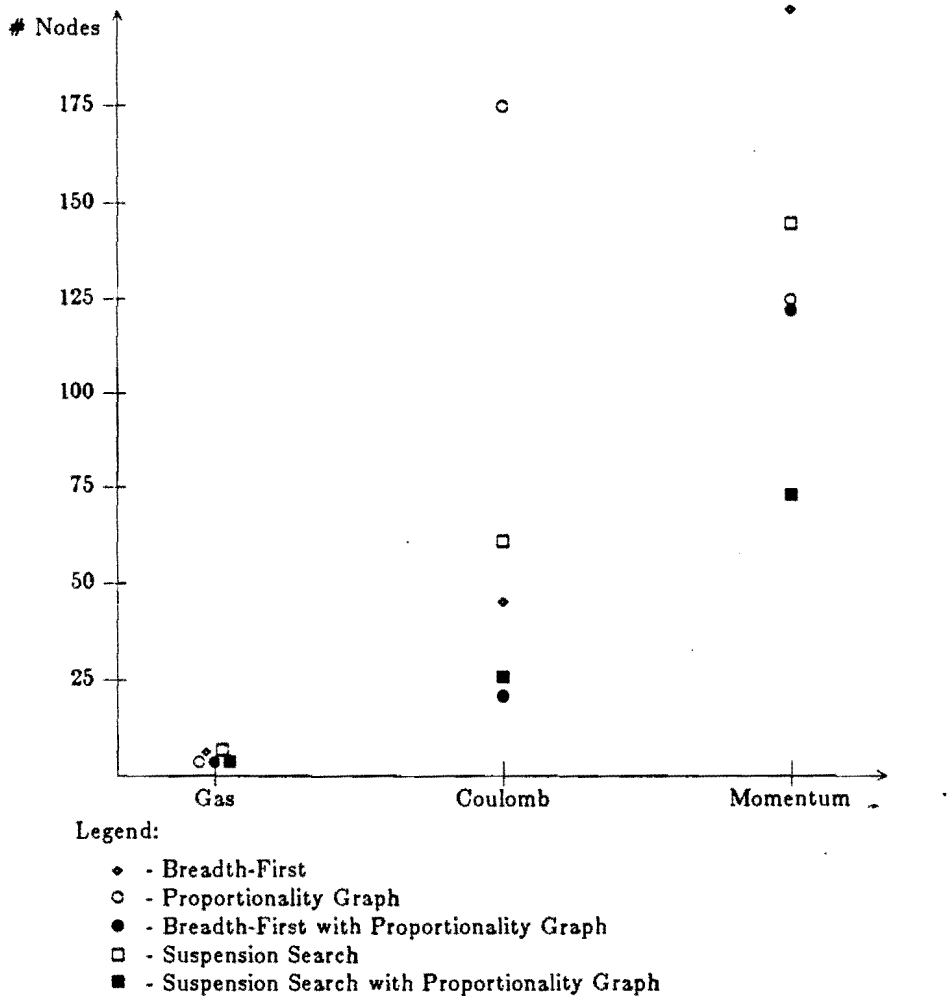


Figure 18. Comparison of Search Algorithms for Different Equation Types

took twice as long but still discovered the relation with a small amount of work.

For the Coulomb example, the problems with the pure proportionality graph approach become apparent. In this particular case, cycle extraction for level two found that every node belonged in the same cycle, leaving it to perform a depth-first search on a cycle set which had 70 edges. Combining the proportionality graph concept with another search method, however, proves to be quite beneficial for this case.

The momentum example differs from the other two in that the relation involved is not a pure product type relation and thus does not lend itself to the heuristics of the proportionality graph strategy. As can be seen, however, using the approach in a first pass only mode does not cause wasteful amounts of search for examples of this type. Finally, the power of the suspension search technique begins to be reflected in the number of nodes generated for this complex example.

3.6. Rewrite and Augmentation Rules

To aid the user in experimentation, rewrite and augmentation rules may be included in the parameters section of the data file. These rules enable one to easily modify an existing variable's values or introduce new variables which are functions of existing variables without having to calculate these values and enter them by hand. If, for example, one wanted to see the affect of converting x to sin(x), the corresponding rewrite rule will simplify this task.

The form of a rewrite rule is:

(<=> variable-name formula units)

Rewrite rules are defined to replace a specified variable's values with those resulting from the given formula which is in terms of existing variables. Thus, given data for attributes x, y, and z, possible rewrite rules would be:

```
(<=> x (sin x) NIL)
(<=> y (times y z) ((meters 2)) )
```

The units are required merely to simplify the parsing task of the rule interpreter. Were they not provided, the rule interpreter in calculating the units itself would have to know the effects all lisp functions (e.g. expt, log, etc.) have on the units of a given variable.

Augmentation rules are used to define a new variable which has values derivable from the other variables. These rules have a similar syntax:

(<= new-variable-name function units)

An augmentation rule will add a new attribute to the original data. Given data for the three attributes x, y, and z, the augmentation rule

(<= w (plus (expt x 2) (times y z)) ((meters 2)))

will add the fourth attribute w to the data and calculate its values in terms of x, y, and z as specified.

3.7. Parameters

Knowledge of the major parameters provided by the ABACUS program will help in understanding better how the system operates. A *constancy factor* may be set to indicate what percentage of the events must be described in order for ABACUS to be able to report a successful discovery. The default value is 40%. The use of the parameter is somewhat complex. As stated earlier, when a constant is found for all of the events or for a nominal subgroup, search terminates immediately. For these cases, the constancy factor is not used. When, however, a partial solution has been found, search continues as long as is permitted by the search depth limit set in the program. For the first three search strategies discussed, namely the non-suspension search algorithms, the constancy factor merely indicates whether ABACUS will report the best equation found after exhausting the search space or whether it will report that no suitable equation could be found. The parameter plays a greater role for the suspension search algorithms. When the search depth limit is reached during the suspension search, the constancy factor is checked. If the best equation found so far has a constancy greater than that set in the parameter, this equation is returned. If not, backtracking commences and suspended nodes are activated until either a suitable solution is found or the search space is examined in full.

A second parameter specifies the uncertainty which will be used during calculations. It is designed to allow for round-off error and noise in the data. The default setting is 2%. Using the default value, when the constancy of a variable is being calculated, numbers falling within a ±2% range of their mean are treated as a constant value. Thus, when ABACUS displays an equation of the form:

$$f(x,y,z) = Constant$$

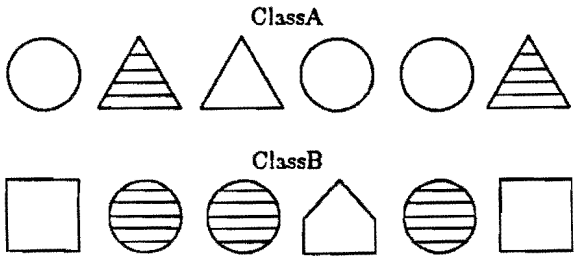
all events described by the equation actually have values within ±2% of *Constant* for the attribute representing that function.

Another parameter signals the proportionality assertion algorithm to ignore certain variables when performing the multi-key sort. If, for example, a nominal variable never has the same value twice, the

proportionality algorithm would be forced to assert prop? at all times were it not told to ignore this attribute. In a previous implementation [Falkenhainer, 1984], a series of settings for this parameter were automatically used if the algorithm failed to discover a satisfactory solution for previous values.

4. DOMAIN CONSTRAINT LEARNING

When multiple equations are discovered for a given set of data, ABACUS associates a logical condition with each equation. These *domain constraints* provide a description of when a particular formula is applicable. Deriving domain constraints for disjoint sets of events is an example of the general covering problem described in [Michalski & Larson, 1978]: Given a list of observed events divided into classes, form a description of each class in terms of the given attributes such that the description describes every event in the class and distinguishes this class from the events in the other classes. This is called a *discriminant description* as it can be used to predict the class a new event should be placed in. For example, suppose we are presented with examples of two classes as in Figure 19(a). Discriminant descriptions of these classes are provided in Figure 19(b). They state that ClassA objects may be uniquely distinguished from objects of ClassB because all objects in ClassA satisfy the logical condition given while none of the objects in ClassB do. The condition for ClassA indicates that objects of this type consist solely of clear circles or any kind of triangle. Similarly, ClassB contains



(a)

ClassA	Cover:	[Object = circle] [Filler = clear] [Object = triangle]
ClassB	Cover:	[Object = circle] [Filler = striped] [Object = square ∨ pentagon]

(b)

Figure 19.

either striped circles or any squares or pentagons. These descriptions were generated by an algorithm known as A^q [Michalski, 1983; Winston, 1984; Becker, 1985a].

4.1. A^q

4.1.1. Definitions

Before embarking on a discussion of the A^q algorithm, a few definitions are needed. The data representation language used by A^q is a variable valued logic known as VL_1 [Michalski & Larson, 1978], an example of which is shown in Figure 19(b). Each term in square brackets specifying the value or values of an individual variable is called a *selector* (e.g. [Object=circle]). A conjunction of selectors, represented by writing them adjacent on a single line, is called a *complex* and forms a partial description of a given class set. The entire description of the class is given by a disjunction of such complexes and is called its *cover*. Thus, VL_1 class descriptions are represented in disjunctive normal form (DNF).

The generalization operator in A^q is ExtendAgainst. To extend selector A against selector B, where A and B represent different selectors for the same attribute, generalize the list of possible values for A without including any values B currently possesses. ExtendAgainst is a selector level operation which returns a list of single selector complexes. When a complex is extended against another complex, the selector for each attribute, x, in the first complex is extended against the corresponding selector for x in the second complex. The result of each selector operation is one or more generalized single selector complexes. Variables in ABACUS may have either linear or nominal domains and the operation is defined differently for each.

ExtendAgainst for nominal variables, an unordered list of ordinal values, is quite simple. To extend instance A of an attribute against instance B is to assign all values in the domain to A, except those values specifically held by B. For the nominal domain {blue, red, green, white, black}, extend against is defined as follows:

Extend([A=blue], [B=redVwhite]) = ([A=blueVgreenVblack])

ExtendAgainst in a linear domain is more complicated. If all values of A are less than B, then A is extended to take on all values in the domain from MIN(domain) to one less than the smallest value of B. Similarly, if all values of A are greater than B, then A takes on values from one greater than B's highest value to MAX(domain). If A's values span those for B, then a list of complexes is generated. As an example, suppose the domain for A and B takes on values between 1 and 10. Then:

Extend([A=2..4], [B=6..7]) = ([A=1..5])
Extend([A=6], [B=2]) = ([A=3..10])
Extend([A=2..10], [B=3..6]) = (([A=1..2]) ([A=7..10]))

At first one might wonder why the selectors are generalized to form complexes on their own, rather than extending the complex as a whole. It is logically more general to extend the single selectors and remove the other constraining conjunctive selectors. As an example, for:

Extend(([x=3] [y=red]), ([x=6] [y=green]))

the complex ([x=1..5]) is more general than the complex ([x=1..5] [y=blue∨red∨white∨black]).

4.1.2. Algorithms

A^q consists primarily of two high-level algorithms. The first is *Cover*, which takes each class set in turn and generates a discriminant cover for the set Figure 20). When a cover is being generated for a class, its events are designated the positive examples and the events from all the other classes are collectively called the negative examples. Cover first selects a single positive event, the *seed*, and passes this to the *Star* algorithm along with the list of negative events. Star will return a list of complexes which represent maximally general descriptions of the seed that do not cover any of the negative events. Cover then selects the best complex according to a user specified lexicographic evaluation function (LEF), adds this disjunctively to the current cover, and removes from the list of positive events those which are described by the new complex. This must necessarily include the seed event. If any events remain uncovered, a new seed is chosen and the process repeats.

The Star algorithm Figure 21) is a beam search through the space of alternate generalized desciptions for the seed. Star generalizes the seed event against each negative event in turn and then through a specialization operation adds this to the current description set (star) such that no previously

```
FUNCTION Cover  (positive_events, negative_events : events) : cover;

VAR
    seed     : event;
    star     : complex_list;
    best     : complex;

BEGIN
  while (positive_events <> nil) do
      seed := head(positive_events);
      star := Star (seed, negative_events);
      best := Bestcomplex (star);
      cover := append (best, cover);
      positive_events := Knockout (positive_events, best);
    end;

END;
```

Figure 20. *Cover* Algorithm

```
FUNCTION Star  (seedevent, negative_events : events)  :  complex_list;

VAR
  genseed : complex_list;

BEGIN
  star := universe;

  for negevent in negative_events do
      genseed := ExtendAgainst (seedevent, negevent);
      star := Multiply (star, genseed, negevent);
      if length(star) > maxstar then
            star := Absorb (star);
            star := SelectBest (star, maxstar, LEF);
          end;
    end;

END;
```

Figure 21. *Star* Algorithm

examined negative events become covered. Examining a single iteration, Star selects a negative event at random and extends the seed description against this negative event, forming a list of alternate

complexes. These complexes are then multiplied into the current star (list of complexes to be returned) to ensure that no description in the star covers the negative event. When the star becomes too large (greater than the specified beam size), it is reduced in size by removing the less desirable complexes as evaluated by the LEF. Once all negative events have been considered in this way, a list of alternate maximally general complexes for describing the seed event remain. Because the star contains generalizations of the seed, it is normally the case that many of the other positive events will also be covered by these descriptions.

4.2. A^q/RU

A modified version of A^q is used in the current ABACUS implementation which I call A^q/RU. It is based on a few of the ideas of [Becker, 1985c] which expanded the theory of A^q to form a new algorithm he uses for exception learning called ExceL. One of the changes made to A^q in ExceL is the replacement of the seed event by the refunion of the positive events. The refunion generalizes the list of positive events as a whole [Michalski, 1980]. This allows the Star algorithm to work with an intermediate description which is more representative of all of the events in a class, rather than trying to generalize a single event in the hope that it alone is representative of the class.

Refunion transforms a set of events into a single complex covering the events. For each variable, the set of all values the variable takes, in all given events, is determined. These sets are used as the reference of the variable in the generated complex. For example, given events of four attributes where the first two attributes (x_1 and x_2) are nominal and the last two (x_3 and x_4) are linear:

$$\begin{aligned} \text{event}_1 &= (2,3,0,1) \\ \text{event}_2 &= (0,2,4,1) \text{ and} \\ \text{event}_3 &= (3,4,0,2) \end{aligned}$$

the refunion complex is:

$$RU(e_1,e_2,e_3) \;=\; [x_1{=}0v2v3][x_2{=}2v3v4][x_3{=}0..4][x_4{=}1..2]$$

Notice that for the nominal attribute x_1, the value of 1 is not included in the refunion while all values between 0 and 4 are included for the linear attribute x_3.

```
FUNCTION Star (refunion : complex; negative_events : events) : complex_list;

VAR
  specialized_ru : complex_list;

BEGIN
  star := universe;

  for negevent in negative_events do
      specialized_ru := Subtract (refunion, negevent);
      star := Multiply (star, specialized_ru, negevent);
      if length(star) > maxstar then
            star := Absorb (star);
            star := SelectBest (star, maxstar, LEF);
          end;
    end;

END;
```

Figure 22. *Star* Algorithm for A^q/RU

Because seed is now being replaced by a generalized description of the class, the generalization step of Star must be replaced with a specialization step (Figure 22). Here it should be observed that Star remains essentially the same except that ExtendAgainst has been replaced by Subtract. Subtract is a selector level operation which returns single selector complexes much like ExtendAgainst, where the values of B intersecting A are removed from A. If A and B represent a linear attribute and a value in the middle of a range is subtracted from A, a list of complexes is returned as for the analogous case in ExtendAgainst.

There are two primary benefits of A^q/RU. First, the refunion generates a more minimal generalization of a class than does ExtendAgainst because it generalizes solely on the values of the positive events. Consequently, the covers generated tend to fit the class more precisely and do not suffer from over-generalization. Second, because the algorithm is working with a more representative description of the entire class from the start, the covers tend to be simpler and more readable than those generated from a seed.

In some instances, the covers generated by A^q/RU are not as good as those generated by A^q. Because A^q/RU starts with a generalized description of each class, disjunctive descriptions are harder to find. For a more detailed discussion see [Becker, 1985c].

4.3. A^q in ABACUS

The covers generated by A^q have two possible uses in the ABACUS system. First, the combination of logical conditions with mathematical equations gives the results a predictive power. Suppose one were able to only obtain values for n-1 variables of an n variable equation. By knowing which equation should apply prior to evaluating it, one could determine the nth attribute from the other n-1 attributes. Second, the logical conditions often provide additional conceptual meaning for the user. For example, Coulomb's law relating the force of attraction of two charge particles separated by a distance r may be stated as

$$\frac{Fr^2}{q_1 q_2} = 4\pi\epsilon$$

where ϵ is defined to be the permittivity of the surrounding medium. Figure 23 shows the results obtained by ABACUS when given measurements for the force (F), the distance (r), the charge of particle one (q_1), the charge of particle two (q_2) and the name of the surrounding medium. From the results it

ClassA	Cover:	[substance=water]
	Relation:	$q_2 * q_1 = 8897.352 * F * r^2$
ClassB	Cover:	[substance=air]
	Relation:	$q_2 * q_1 = 111.280 * F * r^2$
ClassC	Cover:	[substance=silicon]
	Relation:	$q_2 * q_1 = 1312.363 * F * r^2$
ClassD	Cover:	[substance=germanium]
	Relation:	$q_2 * q_1 = 1779.015 * F * r^2$

Figure 23. ABACUS Analysis of Coulomb's Law

can be seen that all of the data obeys the same relationship form where the constant in each case is dependent on the surrounding medium. For this example the domain constraints provide an indication that there is some property associated with each substance which affects the electrical attraction of two charged particles.

5. EXPERIMENTS IN QUANTITATIVE LEARNING

The true test of any piece of work is of course how well it works as intended. Four example experiments will now be presented and discussed to show what types of problems ABACUS can solve. These experiments are:

- The Law of Pendular Motion
- The Law of Conservation of Kinetic Energy
- Gravitational Attraction and Stoke's Law for Viscous Fluids
- A Chemistry Experiment

The first three are experiments designed to illustrate the capabilities of the program. The data used for these was generated by hand with a knowledge of the correct answer. The chemistry experiment represents an original experiment run on data provided by the University of Illinois Chemistry department.

5.1. Pendulum

The motion of a pendulum may be described by the differential equation:

$$\frac{d^2\theta}{dt^2} = \frac{g\sin\theta}{L}$$

where θ is the angle the string makes from the perpendicular, L is the length of the string, and g is the gravitational acceleration downward. If the angle of the pendulum's swing is small compared to the length L, it will very nearly undergo simple harmonic motion. For small angles, the $\sin\theta$ term may be replaced by the close approximation θ and the equation may be solved for the pendulum's period T:

$$T = 2\pi\left(\frac{L}{g}\right)^{1/2}$$

Now let us present ABACUS with some data for different pendulums at various locations where the acceleration g has different values. L is given in meters while the angle θ is given in radians.

As can be seen (Figure 24), ABACUS was able to discover the above relation. It should be noted that the constant 39.48 is the value of $(2\pi)^2$. This final relation, however, did not turn out to equal a

ClassA Cover: [theta= 0.050.. 0.390]
Relation: $T^2 * g = 39.479 * L$

ClassB Cover: [theta= 0.700.. 0.850]
Relation: No formula can be found.

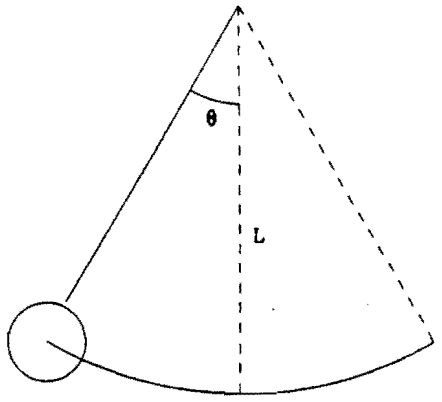


Figure 24. Pendulum Experiment

universal constant for all of the data and no relation could be found for the remaining data.

When A^q/RU examined the above classes, it discovered that the angle θ was the distinctive factor separating these two groups of data. Those data groups which did not hold the above stated relation all had much larger angles. We need only remind ourselves that the relation was intended as an approximation that was valid only when the angle θ was small - a distinction which the ABACUS system discovered.

5.2. Conservation of Momentum and Kinetic Energy

> Law I Every body continues in its state of rest or of uniform motion in a right line, unless it is compelled to change that state by forces impressed upon it.
>
> Issac Newton, Principia, 1686
> translated by Andrew Motte, 1729
> (from [Roller, Blum 1981])

Newton's first law of motion states that the momentum of a body which is not under the influence of a net external force shall remain constant. Generalizing, it may also be stated that the total momentum of two isolated interacting particles will remain constant. Therefore, for two body collisions of the form of Figure 25(a), the vector sum of their individual momenta prior to the collision will equal the vector sum of their momenta after the collision:

(a)

ClassA	Cover:	[collision-type = elastic]
	Relation:	$m_1(v_1^2 - v_1'^2) = m_2(v_2'^2 - v_2^2)$
ClassB	Cover:	[collision-type = inelastic]
	Relation:	No formula can be found

(b)

Figure 25. Conservation of Kinetic Energy in a Two Particle Elastic Collision

$$m_1\mathbf{v}_1 + m_2\mathbf{v}_2 = m_1\mathbf{v}_1' + m_2\mathbf{v}_2'$$

Similarly, the total energy of the system before the time of collision will be the same as the total energy after the collision. For inelastic collisions, some of this energy is converted to heat during the collision and so, at a macroscopic level, an apparent energy loss is observed. For perfectly elastic collisions, however, the sum of the balls' individual energies, namely their kinetic energies, will remain constant before and after their collisions:

$$\tfrac{1}{2}m_1v_1^2 + \tfrac{1}{2}m_2v_2^2 = \tfrac{1}{2}m_1v_1'^2 + \tfrac{1}{2}m_2v_2'^2$$

Converting these concepts into an experiment for ABACUS to examine, data was constructed for a series of observations of various objects colliding. The dataset consisted of 7 attributes and 12 events, where the 7 attributes consisted of the masses of the 2 balls, their 4 corresponding velocities, and a nominal variable which described the observed collision as either elastic or inelastic. The results of

ABACUS's analysis are shown in Figure 25(b).

5.3. Galilean Experiment on Free-Falling Bodies

Now look back to the era when Galileo was hard at work studying the motion of projectiles. He concluded that the flight of all projectiles could be viewed as two completely separate motions, one in a horizontal direction which is unaffected by the pull of the Earth and the other up and down, controlled by the Earth's attraction. His dilemma then was how to describe this vertical component of motion which is so firmly tied to the downward pull of the Earth. His approach was to drop various objects through different fluids and take note of the results. We shall now conduct a similar experiment, aided by modern measurement devices and an ability to create a vacuum which Galileo could not. The balls to be dropped will come in three sizes, for which there will be a rubber ball and a clay ball in each size. The six balls will then be dropped from rest through three different media, namely glycerol, castor oil, and a vacuum, once each for 2 different size containers. The experiment shall be conducted twice, once in Death Valley and once in Denver, and the temperature will be maintained at 20°C at both locations.

Event1:
velocity: 18.064 m/s
radius: 0.05 m
mass: 0.94 kg
time: 0.055 s
height: 1.0 m
substance: Glycerol
location: DeathValley

Event2:
velocity: 6.258 m/s
radius: 0.1 m
mass: 4.985 kg
time: 0.64 s
height: 2.0 m
substance: vacuum
location: Denver

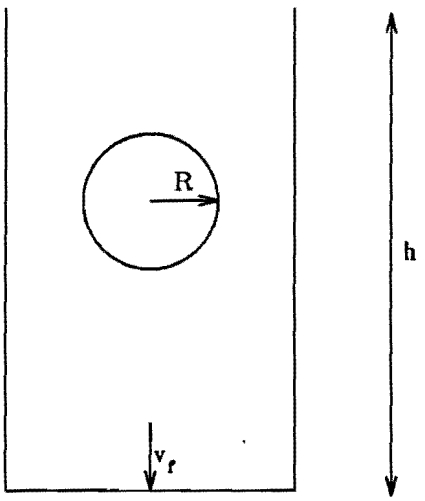


Figure 26. Falling Bodies Experiment

The measured attributes will consist of the height of the container, the mass of the ball, its radius, the duration of the fall, and the velocity with which it strikes the bottom of the container (Figure 26) as measured from the kinetic energy imparted to the base upon impact ($\frac{1}{2}mv^2$). In addition, the substance through which the ball falls will be noted along with the location of the experiment. Samples of the measurements taken are given in Figure 26.[8] Each ball was dropped once through each medium for both containers at each location for a total of 72 events.

When presented with the data, ABACUS discovered the results shown in Figure 27. It appears that for each equation found, its value is dependent upon the medium through which the balls fall and upon the location of the experiment. For classes A and B, it would also appear that the value is independent of the characteristics of the balls used. Interpretting these findings, first we know that, disregarding air friction, an object undergoes a constant acceleration due to earth's gravity and that an object under constant acceleration will change speed proportional to the length of time it undergoes this acceleration.

ClassA	Cover:	[location = DeathValley][substance = Vacuum]
	Relation:	v = 9.8453 * t
ClassB	Cover:	[location = Denver][substance = Vacuum]
	Relation:	v = 9.7898 * t
ClassC	Cover:	[location = DeathValley][substance = Glycerol]
	Relation:	r * v = 0.9583 * m
ClassD	Cover:	[location = Denver][substance = Glycerol]
	Relation:	r * v = 0.9530 * m
ClassE	Cover:	[location = DeathValley][substance = CastorOil]
	Relation:	r * v = 0.7356 * m
ClassF	Cover:	[location = Denver][substance = CastorOil]
	Relation:	r * v = 0.7315 * m

Figure 27. Falling Bodies Experiment Results

[8] The correct measurements were of course calculated by hand for this experiment. The mass of each ball was derived from the chosen radius and the standard densities for rubber and clay. Likewise, the standard viscosity for each substance was used. Gravitational acceleration was chosen to be 9.845 $^{m}/_{s^2}$ in Death Valley and 9.79 $^{m}/_{s^2}$ in Denver to reflect the fact that the force of gravity decreases at higher altitudes.

This may be stated as $\Delta v = a\Delta t$ and corresponds to the cases of the balls falling in a vacuum. The constants for these cases simply represent the earth's gravitational acceleration at the two different locations. When we take the resistance of the medium into account, however, as we must do for glycerol and castor oil, the retarding force of the medium becomes involved and is stated by Stoke's law as:

$$F_r = -6\pi\eta r v$$

where η is the viscosity coefficient of the fluid, r is the radius of the ball, and v is the velocity at some point in time. Because of this added force, the total net force acting on the falling ball now becomes

$$F = mg - 6\pi\eta r v = m\frac{dv}{dt}$$

where g is the gravitational acceleration. Integrating and solving for v, it is found that the object will reach a constant terminal velocity given by

$$v_T = \frac{mg}{6\pi\eta r}$$

Examining the equation reported by ABACUS for the glycerol and castor oil cases, the velocity equation may be converted into the equivalent

$$\frac{v_T r}{m} = \frac{g}{6\pi\eta}$$

It now becomes evident why the values reported for classes C through F were dependent on both the type of liquid (η) and the location (g).

This experiment has shown two important aspects of ABACUS's learning abilities. First, two different equation forms were discovered, each having only the velocity attribute in common. This demonstrates the program's ability to discover multiple equations for different groups of events, even when variables pertinent to one are irrelevant to another. Secondly, the necessity and power of the domain constraint concept can be seen here.

5.4. A Chemistry Experiment

We shall now recreate in part an ongoing chemistry experiment which has been run on authentic experimental data. The problem begins with a desire to predict the distance between the primary metal atoms of bimetallic coordination compounds (Figure 28). These compounds consist of two identical sides,

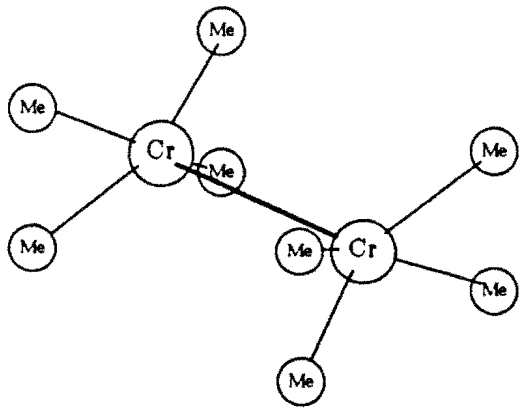


Figure 28. Bimetallic Coordination Compound in Eclipsed Configuration

symmetric about the central covalent bond, each side consisting of a primary metal atom and up to five ligand molecules joined by ionic bond to the primary atom. In experimentation with such molecules, the distance between the metal atoms is difficult and expensive to measure. At present, there is no known way to predict this distance given the values of other attributes and as such, any results presented by the program were completely novel. This provided a unique framework by which to test the usefulness of ABACUS and reveal its strengths and weaknesses. This recreation of the steps taken will also serve to give the reader greater insight into the actual operation of the program.

Data for the experiment consisted of the values of 13 attributes for 30 different observed compounds as garnered from various chemical research articles.[9] The full collection of data is reproduced in Figure 29, where the compound name has been added for thoroughness. "Metal" is the name of the two primary metal atoms, "Ox" is the oxidation state of the metals, "Rad" is the radius of one metal atom in angstroms, and "eM" represents the number of electrons per metal. The bond order of the covalent bond is given by "BO", "MMdist" gives the distance between the metals in angstroms, "Q" is the total charge of the molecule in units of electron charge, and the configuration is shown by "Conf".

[9] Data was collected by Mike Handel and Dr. Ted Brown of the UI Chemistry Department.

Compound	Metal	Ox	MMdist	Rad	Q	eM	Conf	BO	L1	L2	L3	L4	L5
[Cr(CH3)8]4-	Cr	II	1.980	1.29	-4	12	eclip	4	Me	Me	Me	Me	None
[Mo2Cl8]4-	Mo	II	2.139	1.40	-4	12	eclip	4	Cl	Cl	Cl	Cl	None
[Mo(CH3)8]4-	Mo	II	2.148	1.40	-4	12	eclip	4	Me	Me	Me	Me	None
Mo2Br4(p-MeC5H4N)4	Mo	II	2.150	1.40	0	12	eclip	4	Br	NR3	Br	NR3	None
Mo2Cl4(p-MeC5H4N)4	Mo	II	2.153	1.40	0	12	eclip	4	Cl	NR3	Cl	NR3	None
Mo2(CH2SiMe3)6	Mo	III	2.167	1.40	0	9	stag	3	CH2SiR3	CH2SiR3	CH2SiR3	None	None
Mo2(NMe2)4Cl2	Mo	III	2.200	1.40	0	9	stag	3	NMe2	NMe2	Cl	None	None
Mo2(NMe2)4Me2	Mo	III	2.201	1.40	0	9	stag	3	NMe2	NMe2	Me	None	None
Mo2(NMe2)6	Mo	III	2.211	1.40	0	9	stag	3	NMe2	NMe2	NMe2	None	None
Re2Cl6[PEt3]2	Re	III	2.222	1.37	0	12	eclip	4	PEt3	Cl	Cl	Cl	None
Mo2(OCH2CMe3)6	Mo	III	2.222	1.40	0	9	stag	3	OCR3	OCR3	OCR3	None	None
Re2Cl4[P(Et)3]4	Re	II	2.232	1.37	0	14	eclip	4	Cl	PEt3	Cl	PEt3	None
[Re2Cl8]2-	Re	III	2.241	1.37	-2	12	eclip	4	Cl	Cl	Cl	Cl	None
Mo2(OSiMe3)6HNMe2	Mo	III	2.242	1.40	0	11	stag	3	OSiR3	OSiR3	OSiR3	NR3	None
W2[CH2Si(CH3)3]6	W	III	2.255	1.41	0	9	stag	3	CH2SiR3	CH2SiR3	CH2SiR3	None	None
[Re2Br8]2-	Re	III	2.270	1.37	-2	12	eclip	4	Br	Br	Br	Br	None
W2(NMe2)4Cl2	W	III	2.283	1.41	0	9	stag	3	NMe2	NMe2	Cl	None	None
W2(NEt2)4Me2	W	III	2.291	1.41	0	9	stag	3	NEt2	NEt2	Me	None	None
W2(NMe2)6	W	III	2.294	1.41	0	9	stag	3	NMe2	NMe2	NMe2	None	None
W2(NEt2)4I2	W	III	2.296	1.41	0	9	stag	3	NEt2	NEt2	I	None	None
W2Cl2(NEt2)4	W	III	2.301	1.41	0	9	stag	3	Cl	NEt2	NEt2	None	None
W2(NEt2)4Br2	W	III	2.303	1.41	0	9	stag	3	NEt2	NEt2	Br	None	None
W2(OCHMe2)6(pyr)2	W	III	2.332	1.41	0	11	stag	3	OCR3	OCR3	OCR3	NR3	None
Co2(CO)6[P(n-Bu)3]2	Co	O	2.665	1.25	0	17	stag	1	CO	CO	CO	None	PEt3
Mn(CO)8[PEt3]2	Mn	O	2.913	1.37	0	17	stag	1	CO	CO	CO	CO	PEt3
Mn2(CO)10	Mn	O	2.923	1.37	0	17	stag	1	CO	CO	CO	CO	CO
Cr2(CO)10	Cr	-I	2.970	1.29	-2	17	stag	1	CO	CO	CO	CO	CO
Re2(CO)10	Re	O	3.020	1.37	0	17	stag	1	CO	CO	CO	CO	CO
Tc2(CO)10	Tc	O	3.036	1.35	0	17	stag	1	CO	CO	CO	CO	CO
Mo2(CO)10	Mo	-I	3.123	1.40	-2	17	stag	1	CO	CO	CO	CO	CO

Figure 29. Chemistry Experiment Data

In Figure 28, the molecule is in the eclipsed configuration because the ligand molecules of each side line up when viewed from on end. Finally, "L1" through "L5" are the names of the ligand molecules.

The experiment began by running ABACUS using all default parameter values. The response was that no relation could be found. A further examination of the output revealed that no nodes were created, thus indicating that either there were no relations in the data or that all proportionality tests returned prop? and thus no examples could be obtained. To test the later hypothesis, it was concluded that perhaps the large number of nominal variables was interfering with the numerical relation finding

process. As a result, the program was instructed to ignore all nominal variables when trying to hold variables constant for the proportionality test. This time, nodes were created but still no relation was found to hold for the 40% default constancy criterion. As actual measured data might contain a reasonable amount of noise, the decision was made to loosen the default uncertainty of 2% to 5% and then again to 8%. Results now began to be reported. A 40% constancy criterion coupled with a 2% uncertainty was simply too strict. The results of this run are shown in Figure 30, where the number of events in each class is given under the class name. What is most promising about these results is that the same equation was found to hold for all of the data, with only the constant differing. This would indicate the discovery of some type of physical phenomenon more strongly than if a myriad of equations were uncovered, as occurred later when parameters were modified in a variety of ways.

ClassA	Cover:	[L2 = CH2SiR3 v NEt2 v NMe2]
	(13)	[L1 = OCR3][L4 = None]
	Relation:	MMdist = 0.2502 * eM
ClassB	Cover:	[L1 = OSiR3 v PEt3]
	(5)	[L2 = Br]
		[L2 = Cl v OCR3][Metal = Re v W]
	Relation:	MMdist = 0.1954 * eM
ClassC	Cover:	[L2 = CO v Me v NR3 v PEt3]
	(12)	[Ox = II]
	Relation:	MMdist = 0.1735 * eM

Figure 30. Initial Chemistry Results With Uncertainty at 8%

ClassA	Cover:	[Ox = III][L4 = Br v Cl v Me v PEt3 v None]
	Relation:	MMdist * log(BO) = 0.1180 * eM
ClassB	Cover:	[L4 = Me v NR3 v PEt3]
		[Q = -4.0000]
	Relation:	MMdist * log(BO) = 0.1031 * eM

Figure 31. Chemistry Results Using log(BondOrder)

After conducting numerous experiments in this manner, the best results were shown to the members of the chemistry department for their expert opinion. Sadly, while the results looked promising, the conclusion was that these relations did not coincide conceptually with any known physical situation and the uncertainty factor used was far too high for this data. It was suggested, however, to try the logarithm of the bond order as this type of term appeared often in the chemical texts. Continuing with the experimentation, a rewrite rule replacing bond order by log(BondO) was added, a variety of parameter settings were tried, and the conclusions of Figure 31 were obtained. These results look promising as before but again they were not satisfactory from a chemist's perspective and the uncertainty was still too high.

After further analysis of the data by hand, it was reasoned that perhaps there was too much redundancy in the original data. Each metal atom, for example, has a unique radius associated with it. Therefore, the suggestion was made to reduce the number of attributes to 9, consisting of the radius, the metal to metal distance, the bond order, the five ligand names, and a new electrons-per-metal value calculated by adding the bond order to the old value. Since a somewhat new dataset was being used, the uncertainty was returned to its default value of 2% and no variables were to be ignored. On the

ClassA	Cover:	[L1 = CO][Rad = 1.29 .. 1.35] [L1 = CO][Rad = 1.4]
	Relation:	MMdist = 2.2606 * Rad
ClassB	Cover:	[eM* = 18.0]
	Relation:	MMdist = 2.1491 * Rad
ClassC	Cover:	[L1 = Br v Cl v NEt2 v OSiR3 v PEt3][MMdist = 2.222 .. 2.303] [Rad = 1.41]
	Relation:	MMdist = 1.6279 * Rad
ClassD	Cover:	[L2 = CH2SiR3 v Me v NMe2 v NR3][Rad = 1.29 .. 1.4] [L1 = Cl v OCR3][Rad = 1.4]
	Relation:	MMdist = 1.5528 * Rad

Figure 32. Final Chemistry Results Using Reduced Dataset With Uncertainty at 2%

first run, the equations of Figure 32 were obtained. Of significance is the fact that these results represent only a 2% uncertainty. Further experimentation was done using a 1% uncertainty value, but an improvement could not be found. Returning to the first results, the covers were found to all contain selectors for the MMdist attribute. Since the goal is to predict the metal to metal distance, this attribute should not be allowed to appear in the logical condition. By raising the cost of the MMdist attribute, the A^q/RU algorithm was forced to choose another set of covers, specifically those seen in Figure 32. While the attribute was removed from the covers for three of the classes, it still remains for ClassC.

5.5. Analysis

This chapter has attempted to give the reader some insight into the ABACUS quantitative learning system and provide examples of its power and method of operation. A variety of examples have been presented, representing different complexities of equations and domain constraints. The table of Figure 33 shows a comparison of how complex each example turned out to be. Equations composed solely of multiplication and division have been shown to be quite simple learning tasks. This is exhibited in the small number of nodes required for all of the examples except the one for kinetic energy. Even equations of this type possessing powers (Pendulum) appear to be easy to discover. Equations which contain addition or subtraction (Kinetic Energy) are significantly more difficult using the methods

Example	Number of Events	Number of Classes	Total Nodes	Equation Discovery Time	Domain Constraint Time	Total Time
Pendulum	15	2	148	70	2	72
Kinetic Energy	12	2	987	850	5	855
Stoke's Law	72	6	5	18	31	49
Chemistry I	30	3	17	24	16	40
Chemistry III	30	4	5	3	28	31

NOTE: All times are given in CPU seconds

Figure 33. Relevant Statistics of the Quantitative Learning Experiments

presented here, but are still quite manageable. From the times, given in CPU seconds, it can be seen that the program is efficient for every example.

The pendulum example required two exhaustive searches. The first found the desired relation and the second assertained that no formula could be found for those events having a large θ. While the form of the final equation is similar in complexity to that of Stoke's Law, many more nodes were required because of the need to explore the entire search space.

Performance is lessened somewhat by larger numbers of events in the dataset as can be seen by comparing the time required to generate five nodes for the large Stoke's Law dataset against the time required to generate five for the smaller chemistry example. Other factors, such as the number of proportionality tests performed, should be considered when comparing these times.

6. CONCLUSIONS

The theory of equation discovery and domain constraint generation used in ABACUS has been presented and analyzed, and illustrated through a number of examples. ABACUS has proved useful for a number of domains including chemistry, physics, and graph analysis. In addition, it has been shown to be efficient for each learning task.

The work presented here is unique in several ways. First, no prior work has addressed the real-world issue of irrelevant events. Even for cases where the same physical phenomenon appears to be in control, as in the falling bodies example, different physical situations may exist requiring different equation forms to describe them. Previous programs have been unable to discover different equations for different subsets of the given events. Second, the explicit generation of domain constraints for the discovered equations is novel. When different physical situations exist for what appears to be the same phenomenon, domain constraints determine when each equation applies. Finally, a variety of new techniques for the problems of search in equation discovery have been created.

This work has also tried to minimize the amount of information required from the user. Previous systems have required that the user specify which variables are dependent and which are independent. They have also required that all permutations of variable values be given so that prop? assertions will never exist [Langley, 1985]. The later condition means that the user must generate a great many more events that needed here.

Finally, an effort has been made to point out all the issues and difficulties of the problem, some of which proved to be quite subtle during the progress of this work.

6.1. Limitations

There are a number of problems yet unsolved. The possible form of the equations discovered, for example, is limited in this implementation. Equations consist of multiplication, division, addition, and subtraction and they must be of the form

$$f(x) = Constant$$

where *f(z)* is composed solely of user defined variables and operations between them. As a result, general polynomials with coefficients cannot be discovered, thus preventing the discovery of a variety of physical laws. In addition, terms such as sin and log cannot exist unless explicitly created by the user through rewrite or augmentation rules.

Other equation forms cannot be discovered due to the relation finding and equation formation techniques used. For example, data corresponding to parabolas and oscillations often appear to be void of qualitative proportionalities. In addition, incomplete data tends to create a lot of prop? assertions, making the problem more difficult than it would be otherwise.

6.2. Future Research

Of major importance is the recognition of the desired equation once it is created. When the equation applies to all events, the task is easy. When only a percentage of events are covered, when should the search algorithm stop? The problem lies in the basic ignorance of the program. What is needed is some form of conceptual knowledge which would enable it to distinguish between conceptually good equations and conceptually useless equations. This became quite evident during testing of the program when occasionally an unexpected answer would be returned which in fact covered more events than the desired equation, but which was mathematically far more complicated.

The most damaging decision made in this work was to allow irrelevant events in the data. This single decision prevented the use of curve fitting techniques and thus eliminated the possibility of coefficients in the polynomials discovered and prevented or rendered much more difficult the discovery of many very interesting relationships. The search strategy employed and the trend detection algorithm used were forced to be quite loose and as a result the search space was increased and irrelevant variables became harder to locate. A possible solution may be to cluster the events prior to invoking the equation discovery module in some manner such that in each set of events the events all held the same set of proportionality assertions. Given this, the more precise approaches could be taken once again. The method would be based on some form of clustering algorithm, much like the conceptual clustering of [Stepp, 1984]. The clustering algorithm required might possibly be quite simple, merely forming

clustered groups so that all events support the same proportionality assertion.

APPENDIX A: ABACUS.p

ABACUS.p [Falkenhainer 84] was the first implementation of an empirical learning program constructed by the author and the direct predecessor to the current program and theory presented in this thesis. ABACUS.p was implemented in Pascal and runs on a Vax 780. It differed from the current Lisp edition primarily in that it was limited to discovering mathematical laws composed solely of multiplication and division. The assumptions made possible by this limitation drastically eliminate many difficulties and thus have a significant impact on the algorithms involved. As a result, ABACUS.p is very robust and fast in its domain and does not use most of the heuristics and constraints so necessary for the newer version. The primary effect of removing the possibility of addition and subtraction is a redefinition of the search space. Any equation consisting of simply the product and quotient of variables can be derived one variable at a time and so converts the search strategy back to a traditional leaf expansion approach. As a result, ABACUS.p uses a simple depth-first search with a traditional backtracking strategy. While constraints and heuristics might have proved useful, search is so simple under these conditions that no effort was made to construct them. The only constraint present is a check for tautologies as degenerate solutions were often found without it. Because of the assumption present, this test is very inexpensive.

Let us examine the search technique further. Using Figure 34 as a model, we can see that ABACUS.p first created the new variable Fr, leaving m, q_1, q_2 excluded from the current relation. The variable m was next chosen and found to be proportional to Fr, so Frm was created.[1] As m is an irrelevant variable, further search down this subtree will prove to be fruitless. A user supplied parameter states how deep the tree is allowed to go preventing this subtree from being extended as long as proportionalities are still present. As a result, backtracking returns to the Fr node. q_1 is next chosen and the new node Fr/q_1 is created. We can see that, as before, m is tried with no success and so the

[1] The one heuristic present in ABACUS.p is that, from any node, variables excluded from the equation representing that node are tried first. The idea is to include all variables if possible, assuming that most attributes provided by the user are relevant.

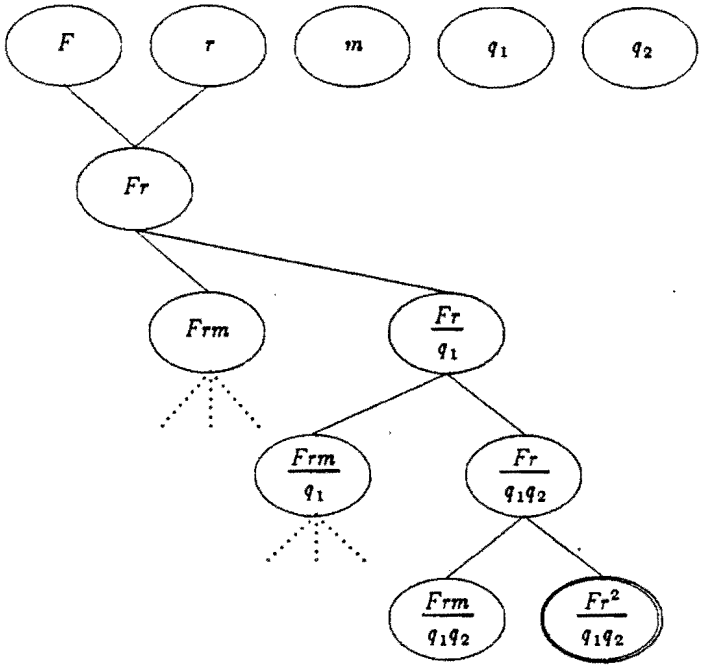


Figure 34. ABACUS.p Search Space for Coulomb's Law

relation Fr/q_1q_2 is formed. Eventually, $\frac{Fr^2}{q_1q_2}$ is discovered, found to hold for a nominal subgroup and so search is terminated. While this search strategy may seem somewhat haphazard, in the end very few nodes are created in comparison to the newer implementation and, since no real constraints or heuristics are involved, node creation is less expensive. Also, ABACUS.p is provided the fact that only one new variable is ever generated for a proportionality assertion between two attributes. This limits the search space further.

An additional benefit of the restriction to multiplication and division and the use of depth-first search is the ability to better define when search should cease. Once a relation is found which holds for a user supplied percentage of the data, backtracking is halted. All possible subtrees desending from the node are examined but, if uneventful, the node is returned as the final solution. If a node further down is found to be superior, then it becomes the new point of no backtracking.

APPENDIX B: ABACUS USER'S GUIDE

ABACUS is a program for experimentation in quantitative empirical learning. It is written in Franz Lisp under the Berkeley UNIX operating system environment. ABACUS is designed to work on a variety of quantitative learning tasks. It has been used to rediscover such laws of Physics and Chemistry as the law of conservation of momentum, the ideal gas law, and Stoke's law for viscous fluids. It has also been used as a laboratory aid in analyzing measurements for bimetallic chemical compounds. Input to ABACUS consists of attribute declarations and a list of observed numerical and symbolic measurements for these attributes. The program generates a clustering of the observed data based on discovered numerical relationships found between the attributes. In addition, logical descriptions are derived for each class to provide a condition on when the numerical relationships are relevant.

B.1. The Dataset

Input to ABACUS consists of a single file divided into four sections. The dataset must be given in list format, appearing as one s-expression to the Lisp read function. The format of the data file is as follows:

```
( (declarations
      <a series of variable declarations>)

  (parameters
      <a series of parameter settings, rewrite rules, and augmentation rules>)

  (variables
      <a list of the variable names>)

  (events
      <the observed data values>)
)
```

The declarations, variables, and events sections must all be present. If the parameters section is missing, all default parameter values will be used. Likewise, parameters not set by the user have defined default values. The order in which the sections are given is unimportant, although the spelling of the four section keywords is. A sample dataset is shown below.

SAMPLE DATASET

```
(
 (declarations
        (P  L  ((kilo-gram 1) (sec -2) (meter -2)))
        (V  L  ((meter 3)))
        (N  L  ((moles 1)))
        (T  L  ((kelvin 1)))
        (STATE N nil)
 )

 (parameters
        (*print-flags* (p s e))
        (*trace* 2)
 )

 (variables
        (V N T P STATE)
 )

 (events
        (4992       42.0        170         0.7         solid)
        (2496       1.0         300         1           gas)
        (1996.8     1.0         300         1.25        gas)
        (1690       20.0        100         2           solid)
        (4992       1.2         250         0.5         gas)
        (4992       0.8         375         0.5         gas)
        (1690       20.0        150         2           solid)
        (4992       1.2         350         0.7         gas)
        (1660       20.0        200         2           solid)
        (4992       1.2         280         0.56        gas)
        (1664       2.0         250         2.5         gas)
        (1664       2.0         350         3.5         gas)
        (1660       20.0        200         1           solid)
        (1660       20.0        210         3           solid)
        (4992       1.2         330         0.66        gas)
        (1660       20.0        220         3.2         solid)
        (4992       1.0         300         0.5         gas)
        (1200       16          110         2.2         solid)
        (1400       17          110         2.2         solid)
))
```

B.1.1. Declarations

The declaration section of the dataset is used to introduce the attributes used. It's structure is given as:

(declarations
  ($variable_1$-name $domain\text{-}type_1$ $units_1$)
  ($variable_2$-name $domain\text{-}type_2$ $units_2$)
  (...)
  (...)
)

Variable names may consist of any valid Lisp symbol whose first character is a letter. A single letter specifies the domain type of the variable and simply indicates whether the variable is numeric or symbolic. Domain type "L" states that the attribute's values are linear (numeric) while "N" states that it's values are nominal (symbolic). Finally, the units field of a declaration tells the program what units the variable has. For symbolic or dimensionless numeric attributes, "nil" should be placed in the units field. Otherwise, the base units of the variable must be given in the following syntax:

(($unit_1$-name $exponent_1$) ($unit_2$-name $exponent_2$) (..) (..))

Base units are those which cannot be expressed in terms of other units. Example basic units are those of mass such as *kilo-grams*, units of time such as *seconds*, or units of distance such as *meters*. A unit of force such as *Newtons* is not a basic unit since it is composed of kilo-grams, seconds, and meters. For example,

(Pressure L ((kg 1) (second -2) (meter -2)))

declares a numeric attribute "Pressure" which has the units $\frac{kg}{m^2 s^2}$.

B.1.2. Parameters

The optional parameter section enables the user to set global parameter values and to declare rewrite and augmentation rules. This section has the following syntax:

(parameters
  (*parameter-name parameter-value*)
  (<= *variable-name evaluation-function units*)
  (<=> *variable-name evaluation-function units*)
  (...)
)

Rewrite Rules

Rewrite rules enable one to easily modify an existing variable's values. The variable's new value

is calculated from a given function which is in terms of the variable's current values and the values of the other attributes. If, for example, one wanted to see the affect of converting x to sin(x), the corresponding rewrite rule will simplify this task. The form of a rewrite rule is:

(<=> *variable-name formula units*)

Rewrite rules are defined to replace a specified variable's values with those resulting from evaluating the given formula. The formula must be given in Lisp notation where any declared variable and any Lisp arithmetic function may be used. Thus, given data for attributes x, y, and z, possible rewrite rules would be:

```
(<=> x (sin x) nil)
(<=> y (times y z) ((meters 2)) )
```

The units are required in order to simplify the parsing task of the rule interpreter. Were they not provided, the rule interpreter in calculating the units itself would have to know the effects all lisp functions (e.g. expt, log, etc.) have on the units of a given variable.

Augmentation Rules

Augmentation rules are used to define a new variable which has values derivable from the other variables. These rules have the syntax:

(<= *new-variable-name formula units*)

An augmentation rule will add a new attribute to the original data. Given data for the three attributes x, y, and z, the augmentation rule

(<= w (plus (expt x 2) (times y z)) ((meters 2)))

will add the fourth attribute w to the data and calculate its values in terms of x, y, and z as specified.

Setting Parameters

A global parameter may be set using the syntax:

(*parameter-name value*)

The user modifiable parameters are given below.

max-nodes (default 1000)

Specifies the maximum number of nodes allowed for any given search tree. If the tree grows to exceed this value, search will terminate and return the best equation found up to that point.

search-limit (default 4)
This parameter applies to the suspension search algorithm. The level number specified by the parameter indicates the level number of the deepest level in the search tree which can be created.

filter-level (default 3)
This parameter applies to the suspension search algorithm. It specifies the deepest level which can contain suspended nodes. Any suspended nodes created at deeper levels are ignored.

univignore (default nil)
Specifies which variables are to be ignored during the proportionality assertion algorithm. The algorithm tries to find a group of events in which all variables are constant except those whose proportionality is being tested. At times, trying to hold certain variables constant prevents ABACUS from being able to draw any conclusions. The variables listed by this parameter are not included in the set of variables which must be constant for one of these groups. When non-nil, the given value should be a list of numbers where the number indicates which column the variable to be ignored corresponds to. Column numbers begin with 0. Thus, (*univignore* (0 3)) states that the variables representing positions 0 and 3 in the event vectors are to be ignored.

direct-sw (default 70) (domain: 0..100)
Indicates the degree of direct proportionality which must exist between two variables for a prop+ assertion to be made. A value of 100 indicates that the values for y must always (100%) rise when the values for x rise for prop+(x,y) to be asserted. A value of 0 will cause the program to always assert that any two variables are directly proportional. Values below 60 should never be specified for this parameter.

inverse-sw (default 30) (domain: 0..100)
Indicates the degree of inverse proportionality which must exist between two variables for a prop- assertion to be made. Conceptually, this parameter maps to the *direct-sw* parameter as:

$$degree\ of\ proportionality = 100 - \text{*inverse-sw*}$$

Thus, a value of 0 for this parameter indicates that the values for y must always decrease when x's values increase for a prop-(x,y) assertion to be made. Value's greater than 40 should never be given for this parameter.

constfactor (default 0.40) (domain: 0..1)
Indicates the percentage of the events which must be constant for an equation to be considered acceptable as a final answer. This only applies to an exhaustive search. When a constant is found for all of the events or for a nominal subgroup, search terminates immediately regardless of the value given here.

uncertainty (default 0.02) (domain: 0..1)
Specifies the uncertainty used during calculations. It is designed to allow for round-off error and noise in the data. Using the default value, when the constancy of a variable is being calculated, numbers falling within a $\pm 2\%$ range of their mean are treated as a constant value. When ABACUS displays an equation of the form:

$$f(x,y,z) = Constant$$

all events described by the equation actually have values within $\pm 2^{\%}$ of *Constant* for the attribute representing that function.

maxstar (default 10)
This parameter indicates the maximum number of complexes allowed in the star for the A^q/RU algorithm. The A^q algorithm is a beam search whose size is indicated by this parameter. When the size of the star exceeds the specified amount, complexes having low values according the evaluation function are discarded from the star until the star is smaller than *maxstar*.

covermode (default ic) (domain: {ic, dc})
Covers generated by the A^q algorithm may be required to be either disjoint or may overlap in *don't care* areas. "ic" covers may overlap while "dc" are required to be disjoint.

starLEF
This parameter provides the list of lexicographic evaluation functions (LEF) which should be used by A^q to trim the size of the current star and to select the best complex of the star. A tolerance is provided with each function, taking on values between 0 and 1, that indicates what percentage of the complexes should be kept as a result of the function's evaluation. If the tolerance is given as 0, then only the best complex, or those complexes tying for the best, will not be trimmed. The pre-defined functions are:

Max-Promise	: % positive events covered / % negative events covered
Min-Covered	: Select complexes covering the fewest events
Max-Covered	: Select complexes covering the most events
Min-Selectors	: Select complexes having the fewest selectors
Max-Selectors	: Select complexes having the most selectors
Min-Weight	: Select complexes of minimum weight by summing selector weights
Max-Weight	: Select complexes of maximum weight by summing selector weights

The default setting for *starLEF* is:
((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))

trace (default 0) (domain: {0, 1, 2})
The tracing facilities may be invoked using this parameter. A value of 0 will turn off all tracing functions. A value of 1 will cause the program to perform only a partial trace. This includes a dump of the search tree after each execution of the equation discovery module and a list of the active and suspended nodes when each level is created. When the trace flag has a value of 2, all tracing functions are invoked. This includes the tracing indicated above as well as the results of all proportionality measurements taken.

print-flags (default (p)) (domain: list of {p, s, e} or nil)
There are three tables which may be optionally output at the end of an ABACUS execution. The *Parameters* table displays all parameter values which were in effect for the run. The *Statistics* table displays various execution time statistics and information about the size of the search trees generated. Finally, the *Events* table shows all of the given events (including the values of variables created from augmentation rules) divided into their final class sets. The parameter may be set to nil or may be a list of key-letters. For example, (*print-flags* (p s e)) states that all three tables should be displayed.

emacs (default nil) (domain: {t, nil})
This flag indicates that the program is being run inside of the emacs editor. When lisp is run inside emacs, screen control commands (e.g. clear screen) are rendered inoperable and cause

"junk" to appear on the screen. This flag suppresses screen controlling I/O.

B.1.3. Variables

The variables section establishes the attribute order for the event vectors. It is simply a list of the variable names declared in the declarations section. The order of the variables given here informs the program which position each variable has in an event vector. Thus

(V N T P STATE)

indicates that for the event vector

(4992 42.0 170 0.7 solid)

V equals 4992, N equals 42, and so on.

B.1.4. Events

The events section is a list of event vectors whose attribute order is defined in the *variables* section. The format of the events section is:

(events
($attribute_0$-$value_1$ $attribute_1$-$value_1$ ... $attribute_{n-1}$-$value_1$)
($attribute_0$-$value_2$ $attribute_1$-$value_2$ ... $attribute_{n-1}$-$value_2$)
(...)
(...)
)

B.2. Execution

ABACUS may be run either interactively or in batch mode. In both cases, the results are the same.

B.2.1. Understanding the Output

Upon completion of a run, ABACUS displays up to four tables, depending on the value of *print-flags*, which summarize the results. The first table displays the parameter values which were in effect:

=-=- Parameters -=-=

maxstar = 10	*covermode* = ic	*uncertainty* = 0.02
constfactor = 0.4	*inverse-sw* = 30	*direct-sw* = 70
univignore = nil	*trace* = 0	*print-flags* = (p s e)
emacs = nil	*filter-level* = 3	*search-limit* = 4
max-nodes = 1000		

starLEF =
((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))

The second table summarizes the statistics about the run. The time required to discover the equation for each class is displayed along with the number of nodes generated for that class's search tree. In addition, the time spent generating the domain constraints for the problem is displayed. All times are in the format *hours:minutes:seconds.thousandths.*

=-=- Statistics -=-=

	Formula Generation			
class	CPU	GC	REAL	Total Nodes
CLASSa	28.200	2.800	37.000	3
CLASSb	2:27.083	2.867	2:30.000	27

Cover Generation
CPU: 8.000
GC: 0.000
REAL: 11.300

The third table displayed simply lists the original events divided into their corresponding class sets.

=-=- Class Events -=-=

CLASSb
(V N T P STATE)
(4992.0 42.0 170.0 0.7 solid)
(4992.0 42.0 160.0 0.6 solid)
(4992.0 42.0 150.0 0.5 solid)
(1660.0 20.0 220.0 3.2 solid)
(1670.0 20.0 210.0 2.0 solid)
(1660.0 20.0 200.0 2.0 solid)
(1660.0 20.0 200.0 1.0 solid)
(1690.0 20.0 160.0 2.0 solid)
(1690.0 20.0 150.0 2.0 solid)
(1690.0 20.0 100.0 2.0 solid)
(1400.0 19.0 200.0 1.0 solid)
(1400.0 17.0 110.0 2.2 solid)
(1200.0 16.0 110.0 2.2 solid)

CLASSa
(V N T P STATE)
(4992.0 0.8 375.0 0.5 gas)
(4992.0 1.2 350.0 0.7 gas)
(1664.0 2.0 350.0 3.5 gas)
(4992.0 1.2 330.0 0.66 gas)
(1996.8 1.0 300.0 1.25 gas)
(2496.0 1.0 300.0 1.0 gas)
(3328.0 1.0 300.0 0.75 gas)
(4992.0 1.0 300.0 0.5 gas)
(4992.0 1.2 250.0 0.5 gas)
(1664.0 2.0 270.0 2.7 gas)
(1664.0 2.0 250.0 2.5 gas)
(4992.0 1.2 280.0 0.56 gas)
(2496.0 1.132 320.0 1.21 gas)

The final conclusions are displayed in the hypotheses table. The number appearing before each complex indicates the number of events covered by that complex.

=-=- Hypotheses -=-=

CLASSa
Cover: 15 [STATE = gas]
Relation: P * V = 8.3202 * N * T

CLASSb
Cover: 15 [STATE = solid]
Relation: No formula can be found.

B.2.2. Interactive Execution

When run interactively, ABACUS presents a menu driven environment. The primary ABACUS menu is shown below.

```
********** ABACUS MENU **********

select one:

   q:  quit
   g:  get and initialize a dataset
   r:  run program
   p:  change program parameters
   f:  output full results to a file
  ab:  run equation-learning program only
  aq:  run Aq (ab or r must have been run at least once)
setq:  setq given atom to dataset

 =>
```

There are currently eight available options. Most of these options are self evident. The "g" option prompts the user for the file name of a desired dataset. The "r" option runs the full ABACUS system on the last dataset loaded. This will first invoke the equation discovery module (*ab*) and then the domain constraint module (*aq*). "setq" is useful for experimentation and debugging. This option will set the internally defined dataset (a list of various structures) to the requested atom. Then using "q" to exit ABACUS and return to the top level Lisp interpretter, one may manipulate the dataset and call various functions directly.

The "p" option invokes the parameters menu:

```
                    =-=- Parameters -=-=

*maxstar* = 10              *covermode* = ic          *uncertainty* = 0.02
*constfactor* = 0.4         *inverse-sw* = 30         *direct-sw* = 70
*univignore* = nil          *trace* = 0               *print-flags* = (p s e)
*emacs* = nil               *filter-level* = 3        *search-limit* = 4
*max-nodes* = 1000

*starLEF* =
        ((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))

Parameter (name or n) =>
```

This menu enables the user to modify any of the global parameters and re-run the program on the current dataset without having to edit the data file and then reloading it. A parameter value may be altered by typing its name at the prompt. The next prompt will ask for the parameter's new value.

Interactive execution requires that the user, once in the Franz Lisp interpretter, load the following two files:

```
-> (load '/ailib/lisplib/util)
-> (load '/mntb/2/michalski/falken/ABACUS/abacus)
```

ABACUS currently exists on the Illinois Department of Computer Science Vax "B" and the Artificial Intelligence Laboratory's Sun Workstation "C". These files are also maintained on tape by the Artificial Intelligence Laboratory's archivist. Once the files are loaded,

```
-> (run)
```

will cause the primary ABACUS menu to appear. After that, option "q" will return the user to the top-level Lisp interpretter.

B.3. Background Execution

Background execution is essentially the same as interactive execution except that all parameter options must be specified in the data file. ABACUS may be run in background by executing from UNIX the commands:

```
% alias abacus "lisp -f /mntb/2/michalski/falken/ABACUS/abacus-bg.o *"
% abacus dataset > output-file &
```

The first line defines the command "abacus" to be the program whose full UNIX pathname is /mntb/2/michalski/falken/ABACUS/abacus-bg.o. This need only be done once each time the user logs on and may be inserted into the user's .login file. Once the alias has been established, the second command line invokes abacus on the given dataset and outputs the results to the given output file.

APPENDIX C: EXPERIMENTAL DATA

C.1. Ideal Gas Law

The data set used for the discovery of the ideal gas law discussed in Chapter 3 is composed of four numerical attributes and the one symbolic attribute indicating the state of the observed substance. When *constfactor* is set high as in this case, ABACUS discovers that the ideal gas law holds when STATE is equal to *gas* and that no formula can be found for STATE equal to solid. When *constfactor* is set to the default value of 40%, various equations are discovered for the STATE events.

Ideal Gas Law Data File

```
    ; Declare all of the variables involved.  A variable of type 'L' is
    ; Linear while a variable of type 'N' is Nominal
((declarations
   (P  L  ((kilo-gram 1) (sec -2) (meter -2)))      ; Declare pressure to be kg/s²m²
   (V  L  ((meter 3)))                              ; Volume in cubic meters
   (N  L  ((moles 1)))                              ; The number of moles
   (T  L  ((kelvin 1)))                             ; The temperature in kelvins
   (STATE N nil)                                    ; The nominal state variable
)

    ; Specify desired parameters
(parameters
   (*constfactor* 0.9)                              ; Provides "No formula found" for solid case
   (*print-flags* (p s e))                          ; Print the parameters, statistics, and events
)

    ; List the variables in the order they will be seen in the events table
(variables
   (V N T P STATE)
)

    ; List the observed events
(events
   (4992     42.0     170      0.7      solid)
   (2496     1.0      300      1        gas)
   (1664     1.0      300      1.5      gas)
   (3328     1.0      300      0.75     gas)
   (1996.8   1.0      300      1.25     gas)
   (1680     20.0     220      2        solid)
   (1690     20.0     100      2        solid)
   (4992     1.2      250      0.5      gas)
   (4992     0.8      375      0.5      gas)
   (1690     20.0     150      2        solid)
   (2496     1.1321   320      1.208    gas)
```

```
(4992    1.2     350     0.7     gas)
(1660    20.0    200     2       solid)
(4992    1.2     280     0.56    gas)
(1664    2.0     310     3.1     gas)
(1664    2.0     250     2.5     gas)
(4992    42.0    150     0.5     solid)
(4992    42.0    160     0.6     solid)
(1664    2.0     270     2.7     gas)
(1664    2.0     350     3.5     gas)
(1660    20.0    200     1       solid)
(1400    19.0    200     1       solid)
(1660    20.0    210     3       solid)
(4992    1.2     330     0.66    gas)
(1660    20.0    220     3.2     solid)
(1670    20.0    210     2       solid)
(4992    1.0     300     0.5     gas)
(1690    20      160     2       solid)
(1200    16      110     2.2     solid)
(1400    17      110     2.2     solid)
))
```

Ideal Gas Law Output

=== Parameters ===

```
*maxstar* = 10              *covermode* = ic          *uncertainty* = 0.02
*constfactor* = 0.9         *inverse-sw* = 30         *direct-sw* = 70
*univignore* = nil          *trace* = 0               *print-flags* = (p s e)
*emacs* = nil               *filter-level* = 2        *search-limit* = 3
*max-nodes* = 1000

*starLEF* =
     ((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))
```

=== Statistics ===

Formula Generation

class	CPU	GC	REAL	Total Nodes
ClassA	2.867	0.000	4.000	3
miscellaneous	26.117	2.367	28.000	28

Cover Generation

CPU:	3.367
GC:	0.000
REAL:	4.000

=== Class Events ===

```
miscellaneous
(V N T P STATE)
(1690.0 20.0 100.0 2.0 solid)
(1200.0 16.0 110.0 2.2 solid)
(1400.0 17.0 110.0 2.2 solid)
(1690.0 20.0 150.0 2.0 solid)
(4992.0 42.0 150.0 0.5 solid)
(1690.0 20.0 160.0 2.0 solid)
```

```
(4992.0 42.0 160.0 0.6 solid)
(4992.0 42.0 170.0 0.7 solid)
(1400.0 19.0 200.0 1.0 solid)
(1660.0 20.0 200.0 1.0 solid)
(1660.0 20.0 200.0 2.0 solid)
(1670.0 20.0 210.0 2.0 solid)
(1660.0 20.0 210.0 3.0 solid)
(1680.0 20.0 220.0 2.0 solid)
(1660.0 20.0 220.0 3.2 solid)

ClassA
(V N T P STATE)
(4992.0 0.8 375.0 0.5 gas)
(4992.0 1.2 350.0 0.7 gas)
(1664.0 2.0 350.0 3.5 gas)
(4992.0 1.2 330.0 0.66 gas)
(1664.0 1.0 300.0 1.5 gas)
(1664.0 2.0 310.0 3.1 gas)
(1996.8 1.0 300.0 1.25 gas)
(2496.0 1.0 300.0 1.0 gas)
(3328.0 1.0 300.0 0.75 gas)
(4992.0 1.0 300.0 0.5 gas)
(4992.0 1.2 250.0 0.5 gas)
(1664.0 2.0 270.0 2.7 gas)
(1664.0 2.0 250.0 2.5 gas)
(4992.0 1.2 280.0 0.56 gas)
(2496.0 1.1321 320.0 1.208 gas)

                    === Hypotheses ===

                          ClassA
  Cover:
       15 [STATE = gas]
  Relation:
          P * V = 8.3202 * N * T

                      miscellaneous
  Cover:
       15 [STATE = solid]
  Relation:
          No formula can be found
```

C.2. Coulomb's Law Data File

The discovery of Coulomb's law shows how domain constraints can sometimes suggest that a nominal variable has an intrinsic property. In this case, the substance type affects the constant in the derived equation. This was discussed in Chapter 4.

Coulomb's Law

```
((declarations
   (substance N nil)                          ; Nominal variable indicating surrounding medium
   (f L ((kg 1) (meter 1) (sec -1)))          ; Force in kg.m/s
   (r L ((meter 1)))                          ; The distance between the particles
   (q1 L ((coulomb 1)))                       ; The charge of each particle in coulombs
   (q2 L ((coulomb 1)))
)

(parameters                                   ; Use all default parameter settings
   (*print-flags* (p s e))
)

(variables
   (substance f q1 q2 r)
)

(events
   (water      1.236      22.000     500.000     1.000)
   (water      1.405      25.000     500.000     1.000)
   (water      2.810      50.000     500.000     1.000)
   (water      2.000     100.090     400.000     1.500)
   (water      0.989      22.000     400.000     1.000)
   (water     20.000      50.000   14235.185     2.000)
   (water      5.000     250.228     400.000     1.500)
   (water      7.025     500.000     500.000     2.000)
   (air        2.000      22.762      22.000     1.500)
   (air        2.000      22.000      22.762     1.500)
   (air       35.555     500.000      31.652     2.000)
   (air       20.000      50.000     100.151     1.500)
   (air       35.555     719.369      22.000     2.000)
   (air       20.000      50.000     178.047     2.000)
   (air       35.555      50.000     178.047     1.500)
   (ice       20.000      25.000    1494.694     2.000)
   (ice        2.000      22.000     169.852     2.000)
   (ice       15.000     560.510      50.000     2.000)
   (ice       20.000     747.347      50.000     2.000)
   (ice       20.000      25.000     840.766     1.500)
   (ice        2.000      22.000     840.766     4.450)
   (ice       15.000      50.000      50.000     0.597)
   (silicon    2.000      22.000     119.300     1.000)
   (silicon   15.000     393.692      50.000     1.000)
   (silicon    1.696      22.000     404.644     2.000)
   (silicon   20.000      50.000    1181.076     1.500)
   (silicon   15.000      98.423      50.000     0.500)
   (silicon    1.696      50.000     178.047     2.000)
   (silicon    1.666      98.423      50.000     1.500)
```

```
  (germanium      1.405     25.000     400.000     2.000)
  (germanium      1.405     50.000     50.000      1.000)
  (germanium      0.702     25.000     50.000      1.000)
  (germanium      1.405     22.000     454.555     2.000)
  (germanium      15.000    150.137    400.000     1.500)
  (germanium      20.000    200.182    400.000     1.500)
  (germanium      20.000    22.000     50.000      0.176)
))
```

Coulomb's Law Output

=== Parameters ===

```
*maxstar* = 10              *covermode* = ic          *uncertainty* = 0.02
*constfactor* = 0.4         *inverse-sw* = 30         *direct-sw* = 70
*univignore* = nil          *trace* = 0               *print-flags* = (p s e)
*emacs* = nil               *filter-level* = 2        *search-limit* = 3
*max-nodes* = 1000

*starLEF* =
     ((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))
```

=== Statistics ===

Formula Generation

class	CPU	GC	REAL	Total Nodes
ClassA	39.417	5.000	1:02.000	25
ClassB	39.417	5.000	1:02.000	25
ClassC	39.417	5.000	1:02.000	25
ClassD	39.417	5.000	1:02.000	25
ClassE	39.417	5.000	1:02.000	25

Cover Generation

```
  CPU:   22.167
  GC:     2.583
  GC:    30.000
```

=== Class Events ===

```
ClassE
(substance f q1 q2 r)
(water 1.236 22.0 500.0 1.0)
(water 0.9890000000000001 22.0 400.0 1.0)
(water 5.0 250.228 400.0 1.5)
(water 20.0 50.0 14235.185 2.0)
(water 2.0 100.09 400.0 1.5)
(water 7.025 500.0 500.0 2.0)
(water 2.81 50.0 500.0 1.0)
(water 1.405 25.0 500.0 1.0)

ClassD
(substance f q1 q2 r)
(germanium 0.702 25.0 50.0 1.0)
(germanium 15.0 150.137 400.0 1.5)
(germanium 1.405 22.0 454.555 2.0)
(germanium 20.0 200.182 400.0 1.5)
```

```
(germanium 1.405 25.0 400.0 2.0)
(germanium 1.405 50.0 50.0 1.0)
(germanium 20.0 22.0 50.0 0.176)

ClassC
(substance f q1 q2 r)
(silicon 1.666 98.423 50.0 1.5)
(silicon 20.0 50.0 1181.076 1.5)
(silicon 15.0 98.423 50.0 0.5)
(silicon 15.0 393.692 50.0 1.0)
(silicon 2.0 22.0 119.3 1.0)
(silicon 1.696 50.0 178.047 2.0)
(silicon 1.696 22.0 404.644 2.0)

ClassB
(substance f q1 q2 r)
(ice 15.0 50.0 50.0 0.597)
(ice 2.0 22.0 169.852 2.0)
(ice 20.0 25.0 840.766 1.5)
(ice 20.0 747.347 50.0 2.0)
(ice 20.0 25.0 1494.694 2.0)
(ice 15.0 560.51 50.0 2.0)
(ice 2.0 22.0 840.766 4.45)

ClassA
(substance f q1 q2 r)
(air 35.555 50.0 178.047 1.5)
(air 2.0 22.762 22.0 1.5)
(air 2.0 22.0 22.762 1.5)
(air 20.0 50.0 178.047 2.0)
(air 35.555 719.369 22.0 2.0)
(air 20.0 50.0 100.151 1.5)
(air 35.555 500.0 31.652 2.0)
```

=== Hypotheses ===

ClassA

Cover:
7 [substance = air]
Relation:
f * r^2 = 0.0090 * q1 * q2

ClassB

Cover:
7 [substance = ice]
Relation:
f * r^2 = 0.0021 * q1 * q2

ClassC

Cover:
7 [substance = silicon]
Relation:
f * r^2 = 0.0008 * q1 * q2

ClassD

Cover:
7 [substance = germanium]
Relation:
f * r^2 = 0.0006 * q1 * q2

ClassE

Cover:
8 [substance = water]
Relation:
f * r^2 = 0.0001 * q1 * q2

C.3. Conservation of Momentum

The conservation of momentum dataset adheres to the non-vector form of the law of conservation of momentum. To complicate the example, the masses change after the collision providing a total of nine attributes.

Conservation of Momentum Data File

```
((declarations
   (M1 L ((kilo-gram 1)))                          ; Masses are in kilogram units
   (V1 L ((meter 1) (sec -1)))                     ; Velocities are in meters per second
   (M2 L ((kilo-gram 1)))
   (V2 L ((meter 1) (sec -1)))
   (M1p L ((kilo-gram 1)))                         ; Mass of object one after collision
   (V1p L ((meter 1) (sec -1)))                    ; Velocity of object one after collision
   (M2p L ((kilo-gram 1)))
   (V2p L ((meter 1) (sec -1)))
   (COLLISION-TYPE N nil)                          ; The nominal collision-type
)

(parameters
   (*constfactor* 0.9)                             ; Cause it to provide 'No formula found' for inelastic
   (*print-flags* (p s e))
)

(variables
   (M1 V1 M2 V2 M1p V1p M2p V2p COLLISION-TYPE)
)

(events
   (10   2    5   2    10   2    5   2    elastic)
   (10   2    5   1    10   1    5   3    elastic)
   (10   2    5   0.5  10   2    10  0.25 elastic)
   (10   1    5   2    20   0.5  5   2    elastic)
   (10   0.5  5   2    7.5  1    15  0.5  elastic)
   (20   0.25 5   2    7.5  1    15  0.5  elastic)
   (20   0.25 10  1    7.5  1    15  0.5  elastic)
   (10   2    5   1    20   0.5  5   3    elastic)
   (10   2    5   1    10   1    7.5 2    elastic)
   (5    5    10  0.5  5    2    10  2    elastic)
   (20   1    10  1    5    2    10  2    elastic)
   (5    5    5   3    10   2    10  2    elastic)
   (5    5    5   3    10   1    15  2    elastic)
   (10   2    5   1    9    1.5  5   0.9  inelastic)
   (5    2    5   2    8    1    4   1.4  inelastic)
   (5    5    10  0.5  4.5  2    9   2    inelastic)
   (20   1    10  1    5.1  2    8.9 2    inelastic)
   (10   1    10  1    9    0.8  9   0.8  inelastic)
   (20   1    15  2    5.1  2    10  2    inelastic)
   (20   5    15  6    20   4.3  10  4.9  inelastic)
   (5    5    10  0.5  5    2    10  0.9  inelastic)
   (10   3    5   4    7.5  2.5  7.5 2    inelastic)
))
```

Conservation of Momentum Output

```
=== Parameters ===

*maxstar* = 10              *covermode* = ic          *uncertainty* = 0.02
*constfactor* = 0.9         *inverse-sw* = 30         *direct-sw* = 70
*univignore* = nil          *trace* = 0               *print-flags* = (p s e)
*emacs* = nil               *filter-level* = 2        *search-limit* = 3
*max-nodes* = 1000

*starLEF* =
     ((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))

=== Statistics ===

                          Formula Generation
class                CPU              GC            REAL      Total Nodes
-------------------------------------------------------------------------
ClassA            58.300           7.400        1:17.000              72
miscellaneous      1.533           0.000           2.000               0

                          Cover Generation
                            CPU:   4.200
                            GC:    0.000
                            GC:    5.910

=== Class Events ===

miscellaneous
(M1 V1 M2 V2 M1p V1p M2p V2p COLLISION-TYPE)
(5.0 2.0 5.0 2.0 8.0 1.0 4.0 1.4 inelastic)
(5.0 5.0 10.0 0.5 4.5 2.0 9.0 2.0 inelastic)
(5.0 5.0 10.0 0.5 5.0 2.0 10.0 0.9 inelastic)
(10.0 3.0 5.0 4.0 7.5 2.5 7.5 2.0 inelastic)
(20.0 1.0 10.0 1.0 5.1 2.0 8.9 2.0 inelastic)
(20.0 1.0 15.0 2.0 5.1 2.0 10.0 2.0 inelastic)
(20.0 5.0 15.0 6.0 20.0 4.3 10.0 4.9 inelastic)
(10.0 1.0 10.0 1.0 9.0 0.8 9.0 0.8 inelastic)
(10.0 2.0 5.0 1.0 9.0 1.5 5.0 0.9 inelastic)

ClassA
(M1 V1 M2 V2 M1p V1p M2p V2p COLLISION-TYPE)
(5.0 5.0 5.0 3.0 10.0 2.0 10.0 2.0 elastic)
(5.0 5.0 5.0 3.0 10.0 1.0 15.0 2.0 elastic)
(10.0 2.0 5.0 2.0 10.0 2.0 5.0 2.0 elastic)
(20.0 1.0 10.0 1.0 5.0 2.0 10.0 2.0 elastic)
(5.0 5.0 10.0 0.5 5.0 2.0 10.0 2.0 elastic)
(10.0 2.0 5.0 1.0 10.0 1.0 5.0 3.0 elastic)
(10.0 2.0 5.0 1.0 10.0 1.0 7.5 2.0 elastic)
(10.0 2.0 5.0 1.0 20.0 0.5 5.0 3.0 elastic)
(10.0 2.0 5.0 0.5 10.0 2.0 10.0 0.25 elastic)
(10.0 1.0 5.0 2.0 20.0 0.5 5.0 2.0 elastic)
(10.0 0.5 5.0 2.0 7.5 1.0 15.0 0.5 elastic)
(20.0 0.25 5.0 2.0 7.5 1.0 15.0 0.5 elastic)
(20.0 0.25 10.0 1.0 7.5 1.0 15.0 0.5 elastic)
```

=-= Hypotheses -=-

ClassA

Cover:
13 [COLLISION-TYPE = elastic]
Relation:
((M2p * V2p) + (M1p * V1p)) = ((M1 * V1) + (M2 * V2))

miscellaneous

Cover:
9 [COLLISION-TYPE = inelastic]
Relation:
No formula can be found

C.4. Pendular Motion

Pendular motion may be described as a function of the period (T), the length of the string (L) and the force of gravity downward (g). The exact law is a second order differential equation, but for small angles (theta), it may be simplified to a simple polynomial. This example shows how ABACUS is able to discern that the angle size is of importance and that attributes not relevant to the mathematical relationship may prove important for the domain constraints.

Pendular Motion Data

```
((declarations
   (l L ((meter 1)))                          ; Length of the supporting string
   (g L ((meter 1) (sec -2)))                 ; Gravitational acceleration in m/s^2
   (t L ((sec 1)))                            ; The observed period of the pendulum
   (theta L nil)                              ; The angle the string makes with the vertical
)

(parameters
   ('print-flags' (p s e))
   ('uncertainty' 0.01)                       ; Need tighter uncertainty to distinguish among thetas
)

(variables
   (l g t theta)
)

(events
   (1.000    9.400    2.049    0.050)
   (2.000    9.200    2.929    0.100)
   (2.000    9.400    2.898    0.200)
   (5.000    9.200    4.632    0.150)
   (5.000    9.800    4.488    0.150)
   (5.000    9.800    4.488    0.050)
   (5.000    9.400    4.583    0.230)
   (5.000    9.400    4.583    0.280)
   (2.000    9.800    2.839    0.390)
   (1.000    9.800    2.007    0.390)
   (5.000    9.200    4.633    0.220)
   (5.000    9.800    4.650    0.700)
   (2.000    9.400    3.010    0.700)
   (2.000    9.200    3.090    0.850)
   (2.000    9.800    3.000    0.800)
))
```

Pendular Motion Output

```
                          == Parameters ==

*maxstar* = 10              *covermode* = ic          *uncertainty* = 0.01
*constfactor* = 0.4         *inverse-sw* = 30         *direct-sw* = 70
*univignore* = nil          *trace* = 0               *print-flags* = (p s e)
*emacs* = nil               *filter-level* = 2        *search-limit* = 3
*max-nodes* = 1000

*starLEF* =
     ((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))

                          == Statistics ==

                          Formula Generation
class                     CPU            GC          REAL      Total Nodes
-------------------------------------------------------------------------
ClassA              1:09.983        10.350       1:14.000          148
miscellaneous         0.200         0.000          0.000            0

                          Cover Generation
                            CPU:        1.467
                            GC:         0.000
                            REAL:       2.000

                          == Class Events ==

miscellaneous
(l g t theta)
(2.0 9.2 3.09 0.85)
(2.0 9.4 3.01 0.7)
(2.0 9.8 3.0 0.8)
(5.0 9.8 4.65 0.7)

ClassA
(l g t theta)
(2.0 9.2 2.929 0.1)
(1.0 9.4 2.049 0.05)
(2.0 9.4 2.898 0.2)
(1.0 9.8 2.007 0.39)
(5.0 9.2 4.632 0.15)
(5.0 9.8 4.488 0.05)
(5.0 9.8 4.488 0.15)
(5.0 9.4 4.583 0.28)
(5.0 9.4 4.583 0.23)
(2.0 9.8 2.839 0.39)
(5.0 9.2 4.633 0.22)

                          == Hypotheses ==

                                ClassA
   Cover:
         11 [theta = 0.05 .. 0.39]
   Relation:
            g * t^2 = 39.4795 * l

                            miscellaneous
   Cover:
         4  [theta = 0.7 .. 0.85]
   Relation:
            No formula can be found
```

C.5. Kinetic Energy

This is the standard form of the law of conservation of kinetic energy for elastic collisions. There are seven attributes, one indicating whether the observed collision was elastic or inelastic.

Kinetic Energy Data File

```
((declarations
   (M1 L ((kilo-gram 1)))
   (V1 L ((meter 1) (sec -1)))                    ; Velocity of object one before collision
   (M2 L ((kilo-gram 1)))
   (V2 L ((meter 1) (sec -1)))
   (V1p L ((meter 1) (sec -1)))                   ; Velocity of object one after collision
   (V2p L ((meter 1) (sec -1)))
   (COLLISION-TYPE N nil)
)

(parameters
   ("constfactor" 0.9)                            ; Cause it to provide 'no formula found' for inelastic
   ("print-flags" (p s e))
)

(variables
   (M1 V1 V1p M2 V2 V2p COLLISION-TYPE)
)

(events
   (5      3       2       5       2       3       elastic)
   (5      5       4.4721  5       2       3       elastic)
   (5      3       2       5       4.4721  5       elastic)
   (10     3       2       5       6       6.7823  elastic)
   (10     3       2       10      2       3       elastic)
   (5      6.7823  6       10      2       3       elastic)
   (20     1       10      1       2       2       inelastic)
   (10     1       10      1       0.8     0.8     inelastic)
   (20     1       15      2       2       2       inelastic)
   (20     5       15      6       4.3     4.9     inelastic)
   (5      5       10      0.5     2       0.9     inelastic)
   (10     3       5       4       2.5     2       inelastic)
))
```

Kinetic Energy Output

```
                    =-= Parameters -=-

*maxstar* = 10              *covermode* = ic          *uncertainty* = 0.02
*constfactor* = 0.9         *inverse-sw* = 30         *direct-sw* = 70
*univignore* = nil          *trace* = 0               *print-flags* = (p s e)
*emacs* = nil               *filter-level* = 2        *search-limit* = 3
*max-nodes* = 2000

*starLEF* =
     ((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))

                    =-= Statistics -=-

                          Formula Generation
class                CPU              GC            REAL      Total Nodes
-------------------------------------------------------------------------
ClassA          10:14.067        5:35.867       29:48.000          712
ClassB           3:56.067        1:35.867       17:12.000          275

                          Cover Generation
                            CPU:          4.850
                            GC:           0.000
                            REAL:         6.000

                    =-= Class Events -=-
miscellaneous
(M1 V1 M2 V2 V1p V2p COLLISION-TYPE)
(20 1 10 1 2 2 inelastic)
(20 1 15 2 2 2 inelastic)
(20 5 15 6 4.3 4.9 inelastic)
(10 1 10 1 0.8 0.8 inelastic)
(10 3 5 4 2.5 2 inelastic)
(5 5 10 0.5 2 0.9 inelastic)

ClassA
(M1 V1 M2 V2 V1p V2p COLLISION-TYPE)
(5.0 5.0 5.0 2.0 3.0 4.4721 elastic)
(10.0 3.0 10.0 2.0 3.0 2.0 elastic)
(10.0 3.0 5.0 6.0 6.78233 2.0 elastic)
(5.0 6.78233 10.0 2.0 3.0 6.0 elastic)
(5.0 3.0 5.0 4.4721 5.0 2.0 elastic)
(5.0 3.0 5.0 2.0 3.0 2.0 elastic)

                    =-= Hypotheses -=-

                              ClassA
   Cover:
         6   [COLLISION-TYPE = elastic]
   Relation:
            M1 * (V1^2 - V1p^2) = M2 * (V2p^2 - V2^2)

                          miscellaneous
   Cover:
         6   [COLLISION-TYPE = inelastic]
   Relation:
            No formula can be found
```

C.6. Falling Bodies and Stoke's Law

This dataset represents data generated to recreate an imaginary experiment in which 6 balls are dropped through three different mediums for two different height containers at two locations. Three balls are composed of rubber (density 1.19 g/cm^3) and three are composed of clay (density 1.8 g/cm^3). The viscosity of glycerol at 25° C is taken to be 545 × 10^{-3} Nm/s^2 while the viscosity of castor oil is 710 × 10^{-3} Nm/s^2. The acceleration due to gravity (g) at Death Valley was arbitrarily chosen to be 9.845 m/s^2 while g at Denver was chosen to be 9.79 m/s^2. These values were chosen to reflect the fact that g is less at higher altitudes. ABACUS was instructed to ignore variables 3 (t) and 4 (h) because, for the vacuum case, the program cannot hold the height constant while comparing the velocity to the time of fall (time does not change for a given height). Similarly, for the glycerol and castor oil cases, the time could not be held constant while comparing the velocity to the radius or to the mass (each of the six balls has a unique time). The dataset used in chapter 5 is shown below:

Falling Bodies Data

```
((declarations
   (v  L  ((meter 1) (sec -1)))      ; Final velocity of the ball
   (r  L  ((meter 1)))               ; The radius of the ball
   (m  L  ((kg 1)))                  ; Mass of the ball in kilograms
   (t  L  ((sec 1)))                 ; The time required to fall to the container bottom
   (h  L  ((meter 1)))               ; The height of the container
   (substance N nil)                 ; The substance through which the ball fell
   (location N nil)                  ; The location which corresponds to a difference in gravity
 )

 (parameters
   ("univignore" (3 4))              ; Ignore t and h for proportionality assertions
   ("uncertainty" 0.002)             ; Tighten down uncertainty
   ("print-flags" (p s e))
 )

 (variables
   (v r m t h substance location)
 )

 (events
   (11.9425    0.05    0.6231    0.0837    1.0    Glycerol    DeathValley)
   (26.8705    0.075   2.1029    0.0372    1.0    Glycerol    DeathValley)
   (47.7698    0.1     4.9847    0.0209    1.0    Glycerol    DeathValley)
   (18.0642    0.05    0.9425    0.0554    1.0    Glycerol    DeathValley)
```

(40.6445	0.075	3.1809	0.0246	1.0	Glycerol	DeathValley)
(72.2569	0.1	7.5398	0.0138	1.0	Glycerol	DeathValley)
(9.1671	0.05	0.6231	0.1091	1.0	CastorOil	DeathValley)
(20.6260	0.075	2.1029	0.0485	1.0	CastorOil	DeathValley)
(36.6684	0.1	4.9847	0.0273	1.0	CastorOil	DeathValley)
(13.8662	0.05	0.9425	0.0721	1.0	CastorOil	DeathValley)
(31.1989	0.075	3.1809	0.0321	1.0	CastorOil	DeathValley)
(55.4648	0.1	7.5398	0.0180	1.0	CastorOil	DeathValley)
(11.8757	0.05	0.6231	0.0842	1.0	Glycerol	Denver)
(26.7204	0.075	2.1029	0.0374	1.0	Glycerol	Denver)
(47.5030	0.1	4.9847	0.0211	1.0	Glycerol	Denver)
(17.9633	0.05	0.9425	0.0557	1.0	Glycerol	Denver)
(40.4174	0.075	3.1809	0.0247	1.0	Glycerol	Denver)
(71.8532	0.1	7.5398	0.0139	1.0	Glycerol	Denver)
(9.1159	0.05	0.6231	0.1097	1.0	CastorOil	Denver)
(20.5107	0.075	2.1029	0.0488	1.0	CastorOil	Denver)
(36.4635	0.1	4.9847	0.0274	1.0	CastorOil	Denver)
(13.7887	0.05	0.9425	0.0725	1.0	CastorOil	Denver)
(31.0246	0.075	3.1809	0.0322	1.0	CastorOil	Denver)
(55.1549	0.1	7.5398	0.0181	1.0	CastorOil	Denver)
(11.9425	0.05	0.6231	0.1675	2.0	Glycerol	DeathValley)
(26.8705	0.075	2.1029	0.0744	2.0	Glycerol	DeathValley)
(47.7698	0.1	4.9847	0.0419	2.0	Glycerol	DeathValley)
(18.0642	0.05	0.9425	0.1107	2.0	Glycerol	DeathValley)
(40.6445	0.075	3.1809	0.0492	2.0	Glycerol	DeathValley)
(72.2569	0.1	7.5398	0.0277	2.0	Glycerol	DeathValley)
(9.1671	0.05	0.6231	0.2182	2.0	CastorOil	DeathValley)
(20.6260	0.075	2.1029	0.0970	2.0	CastorOil	DeathValley)
(36.6684	0.1	4.9847	0.0545	2.0	CastorOil	DeathValley)
(13.8662	0.05	0.9425	0.1442	2.0	CastorOil	DeathValley)
(31.1989	0.075	3.1809	0.0641	2.0	CastorOil	DeathValley)
(55.4648	0.1	7.5398	0.0361	2.0	CastorOil	DeathValley)
(11.8757	0.05	0.6231	0.1684	2.0	Glycerol	Denver)
(26.7204	0.075	2.1029	0.0748	2.0	Glycerol	Denver)
(47.5030	0.1	4.9847	0.0421	2.0	Glycerol	Denver)
(17.9633	0.05	0.9425	0.1113	2.0	Glycerol	Denver)
(40.4174	0.075	3.1809	0.0495	2.0	Glycerol	Denver)
(71.8532	0.1	7.5398	0.0278	2.0	Glycerol	Denver)
(9.1159	0.05	0.6231	0.2194	2.0	CastorOil	Denver)
(20.5107	0.075	2.1029	0.0975	2.0	CastorOil	Denver)
(36.4635	0.1	4.9847	0.0548	2.0	CastorOil	Denver)
(13.7887	0.05	0.9425	0.1450	2.0	CastorOil	Denver)
(31.0246	0.075	3.1809	0.0645	2.0	CastorOil	Denver)
(55.1549	0.1	7.5398	0.0363	2.0	CastorOil	Denver)
(4.4373	0.05	0.6231	0.4507	1.0	Vacuum	DeathValley)
(4.4373	0.075	2.1029	0.4507	1.0	Vacuum	DeathValley)
(4.4373	0.1	4.9847	0.4507	1.0	Vacuum	DeathValley)
(4.4373	0.05	0.9425	0.4507	1.0	Vacuum	DeathValley)
(4.4373	0.075	3.1809	0.4507	1.0	Vacuum	DeathValley)
(4.4373	0.1	7.5398	0.4507	1.0	Vacuum	DeathValley)
(4.4249	0.05	0.6231	0.4520	1.0	Vacuum	Denver)
(4.4249	0.075	2.1029	0.4520	1.0	Vacuum	Denver)
(4.4249	0.1	4.9847	0.4520	1.0	Vacuum	Denver)
(4.4249	0.05	0.9425	0.4520	1.0	Vacuum	Denver)
(4.4249	0.075	3.1809	0.4520	1.0	Vacuum	Denver)
(4.4249	0.1	7.5398	0.4520	1.0	Vacuum	Denver)
(6.2753	0.05	0.6231	0.6374	2.0	Vacuum	DeathValley)
(6.2753	0.075	2.1029	0.6374	2.0	Vacuum	DeathValley)
(6.2753	0.1	4.9847	0.6374	2.0	Vacuum	DeathValley)
(6.2753	0.05	0.9425	0.6374	2.0	Vacuum	DeathValley)
(6.2753	0.075	3.1809	0.6374	2.0	Vacuum	DeathValley)
(6.2753	0.1	7.5398	0.6374	2.0	Vacuum	DeathValley)
(6.2578	0.05	0.6231	0.6392	2.0	Vacuum	Denver)
(6.2578	0.075	2.1029	0.6392	2.0	Vacuum	Denver)
(6.2578	0.1	4.9847	0.6392	2.0	Vacuum	Denver)
(6.2578	0.05	0.9425	0.6392	2.0	Vacuum	Denver)

```
  (6.2578       0.075      3.1809      0.6392      2.0       Vacuum       Denver)
  (6.2578       0.1        7.5398      0.6392      2.0       Vacuum       Denver)
))
```

Falling Bodies Output

```
                              === Parameters ===

  *maxstar* = 10              *covermode* = ic            *uncertainty* = 0.002
  *constfactor* = 0.4         *inverse-sw* = 30           *direct-sw* = 70
  *univignore* = (3 4)        *trace* = 0                 *print-flags* = (p s e)
  *emacs* = nil               *filter-level* = 2          *search-limit* = 3
  *max-nodes* = 1000

  *starLEF* =
       ((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))

                              === Statistics ===

                              Formula Generation
  class                  CPU              GC              REAL        Total Nodes
  ------------------------------------------------------------------------------
  ClassA                 8.283            0.000           15.000            2
  ClassB                 8.283            0.000           15.000            2
  ClassC                 9.133            0.000           13.000            3
  ClassD                 9.133            0.000           13.000            3
  ClassE                 9.133            0.000           13.000            3
  ClassF                 9.133            0.000           13.000            3

                              Cover Generation
                                CPU:          30.517
                                GC:            5.400
                                REAL:         38.000

                              === Class Events ===

ClassF
(v r m t h substance location)
(9.1159 0.05 0.6231 0.2194 2.0 CastorOil Denver)
(9.1159 0.05 0.6231 0.1097 1.0 CastorOil Denver)
(13.7887 0.05 0.9425000000000001 0.145 2.0 CastorOil Denver)
(13.7887 0.05 0.9425000000000001 0.0725 1.0 CastorOil Denver)
(31.0246 0.075 3.1809 0.0645 2.0 CastorOil Denver)
(31.0246 0.075 3.1809 0.0322 1.0 CastorOil Denver)
(36.4635 0.1 4.9847 0.0274 1.0 CastorOil Denver)
(36.4635 0.1 4.9847 0.0548 2.0 CastorOil Denver)
(20.5107 0.075 2.1029 0.09750000000000002 2.0 CastorOil Denver)
(20.5107 0.075 2.1029 0.04880000000000001 1.0 CastorOil Denver)
(55.1549 0.1 7.5398 0.0181 1.0 CastorOil Denver)
(55.1549 0.1 7.5398 0.0363 2.0 CastorOil Denver)

ClassE
(v r m t h substance location)
(9.1671 0.05 0.6231 0.1091 1.0 CastorOil DeathValley)
```

```
(9.1671 0.05 0.6231 0.2182 2.0 CastorOil DeathValley)
(13.8662 0.05 0.9425000000000001 0.0721 1.0 CastorOil DeathValley)
(13.8662 0.05 0.9425000000000001 0.1442 2.0 CastorOil DeathValley)
(31.1989 0.075 3.1809 0.0321 1.0 CastorOil DeathValley)
(31.1989 0.075 3.1809 0.0641 2.0 CastorOil DeathValley)
(36.6684 0.1 4.9847 0.0545 2.0 CastorOil DeathValley)
(36.6684 0.1 4.9847 0.0273 1.0 CastorOil DeathValley)
(55.4648 0.1 7.5398 0.0361 2.0 CastorOil DeathValley)
(55.4648 0.1 7.5398 0.018 1.0 CastorOil DeathValley)
(20.626 0.075 2.1029 0.09700000000000001 2.0 CastorOil DeathValley)
(20.626 0.075 2.1029 0.04850000000000001 1.0 CastorOil DeathValley)

ClassD
(v r m t h substance location)
(11.8757 0.05 0.6231 0.1684 2.0 Glycerol Denver)
(11.8757 0.05 0.6231 0.0842 1.0 Glycerol Denver)
(17.9633 0.05 0.9425000000000001 0.0557 1.0 Glycerol Denver)
(17.9633 0.05 0.9425000000000001 0.1113 2.0 Glycerol Denver)
(40.4174 0.075 3.1809 0.0247 1.0 Glycerol Denver)
(40.4174 0.075 3.1809 0.0495 2.0 Glycerol Denver)
(47.503 0.1 4.9847 0.0421 2.0 Glycerol Denver)
(47.503 0.1 4.9847 0.0211 1.0 Glycerol Denver)
(26.7204 0.075 2.1029 0.07480000000000001 2.0 Glycerol Denver)
(26.7204 0.075 2.1029 0.03740000000000001 1.0 Glycerol Denver)
(71.8532 0.1 7.5398 0.0277 2.0 Glycerol Denver)
(71.8532 0.1 7.5398 0.0139 1.0 Glycerol Denver)

ClassC
(v r m t h substance location)
(18.0642 0.05 0.9425000000000001 0.1107 2.0 Glycerol DeathValley)
(18.0642 0.05 0.9425000000000001 0.05539999999999999 1.0 Glycerol DeathValley)
(11.9425 0.05 0.6231 0.1675 2.0 Glycerol DeathValley)
(11.9425 0.05 0.6231 0.08369999999999999 1.0 Glycerol DeathValley)
(40.6445 0.075 3.1809 0.0492 2.0 Glycerol DeathValley)
(40.6445 0.075 3.1809 0.0246 1.0 Glycerol DeathValley)
(47.7698 0.1 4.9847 0.0419 2.0 Glycerol DeathValley)
(47.7698 0.1 4.9847 0.0209 1.0 Glycerol DeathValley)
(26.8705 0.075 2.1029 0.0744 2.0 Glycerol DeathValley)
(26.8705 0.075 2.1029 0.0372 1.0 Glycerol DeathValley)
(72.2569 0.1 7.5398 0.0277 2.0 Glycerol DeathValley)
(72.2569 0.1 7.5398 0.0138 1.0 Glycerol DeathValley)

ClassB
(v r m t h substance location)
(4.4249 0.1 7.5398 0.452 1.0 Vacuum Denver)
(4.4249 0.075 3.1809 0.452 1.0 Vacuum Denver)
(4.4249 0.1 4.9847 0.452 1.0 Vacuum Denver)
(4.4249 0.05 0.9425000000000001 0.452 1.0 Vacuum Denver)
(4.4249 0.075 2.1029 0.452 1.0 Vacuum Denver)
(4.4249 0.05 0.6231 0.452 1.0 Vacuum Denver)
(6.2578 0.1 7.5398 0.6392 2.0 Vacuum Denver)
(6.2578 0.1 4.9847 0.6392 2.0 Vacuum Denver)
(6.2578 0.075 3.1809 0.6392 2.0 Vacuum Denver)
(6.2578 0.075 2.1029 0.6392 2.0 Vacuum Denver)
(6.2578 0.05 0.9425000000000001 0.6392 2.0 Vacuum Denver)
(6.2578 0.05 0.6231 0.6392 2.0 Vacuum Denver)

ClassA
(v r m t h substance location)
(6.2753 0.1 7.5398 0.6374 2.0 Vacuum DeathValley)
(6.2753 0.1 4.9847 0.6374 2.0 Vacuum DeathValley)
(6.2753 0.075 3.1809 0.6374 2.0 Vacuum DeathValley)
(6.2753 0.075 2.1029 0.6374 2.0 Vacuum DeathValley)
(6.2753 0.05 0.9425000000000001 0.6374 2.0 Vacuum DeathValley)
(6.2753 0.05 0.6231 0.6374 2.0 Vacuum DeathValley)
(4.4373 0.1 7.5398 0.4507 1.0 Vacuum DeathValley)
```

```
(4.4373 0.1 4.9847 0.4507 1.0 Vacuum DeathValley)
(4.4373 0.075 3.1809 0.4507 1.0 Vacuum DeathValley)
(4.4373 0.075 2.1029 0.4507 1.0 Vacuum DeathValley)
(4.4373 0.05 0.9425000000000001 0.4507 1.0 Vacuum DeathValley)
(4.4373 0.05 0.6231 0.4507 1.0 Vacuum DeathValley)
```

```
=== Hypotheses ===

                    ClassA
Cover:
      12 [location = DeathValley][substance = Vacuum]
Relation:
         v = 9.8453 * t

                    ClassB
Cover:
      12 [location = Denver][substance = Vacuum]
Relation:
         v = 9.7898 * t

                    ClassC
Cover:
      12 [location = DeathValley][substance = Glycerol]
Relation:
         r * v = 0.9583 * m

                    ClassD
Cover:
      12 [location = Denver][substance = Glycerol]
Relation:
         r * v = 0.9530 * m

                    ClassE
Cover:
      12 [location = DeathValley][substance = CastorOil]
Relation:
         r * v = 0.7356 * m

                    ClassF
Cover:
      12 [location = Denver][substance = CastorOil]
Relation:
         r * v = 0.7315 * m
```

In the above dataset, one should notice that the uncertainty is set at $0.2^{\%}$. This low uncertainty was required in order to distinguish between the two locations (Death Valley and Denver). If the default uncertainty of $2^{\%}$ is used, the following results are obtained:

Stoke's Data With 2% Uncertainty

=== Parameters ===

```
*maxstar* = 10              *covermode* = ic          *uncertainty* = 0.02
*constfactor* = 0.4         *inverse-sw* = 30         *direct-sw* = 70
*univignore* = (3 4)        *trace* = 0               *print-flags* = (p s)
*emacs* = nil               *filter-level* = 2        *search-limit* = 3
*max-nodes* = 1000

*starLEF* =
     ((Max-Covered 0.5) (Min-Weight 0.0) (Min-Selectors 0.0))
```

=== Statistics ===

Formula Generation

class	CPU	GC	REAL,	Total Nodes
CLASSa	8.050	0.000	21.000	2
CLASSb	8.933	0.000	11.000	3
CLASSc	8.933	0.000	11.000	3

Cover Generation

CPU:	16.850
GC:	2.750
REAL:	24.000

=== Hypotheses ===

CLASSa

```
Cover:
      24 [substance = Vacuum]
Relation:
         v = 9.8175 * t
```

CLASSb

```
Cover:
      24 [substance = Glycerol]
Relation:
         r * v = 0.9556 * m
```

CLASSc

```
Cover:
      24 [substance = CastorOil]
Relation:
         r * v = 0.7336 * m
```

C.7. Chemistry Data

This data was collected from the University of Illinois Chemistry department and is described in detail in Chapter 5.

Compound	Metal	Ox	MMdist	Rad	Q	eM	Conf	BO	L1	L2	L3	L4	L5
[Cr(CH3)8]4-	Cr	II	1.980	1.29	-4	12	eclip	4	Me	Me	Me	Me	None
[Mo2Cl8]4-	Mo	II	2.139	1.40	-4	12	eclip	4	Cl	Cl	Cl	Cl	None
[Mo(CH3)8]4-	Mo	II	2.148	1.40	-4	12	eclip	4	Me	Me	Me	Me	None
Mo2Br4(p-MeC5H4N)4	Mo	II	2.150	1.40	0	12	eclip	4	Br	NR3	Br	NR3	None
Mo2Cl4(p-MeC5H4N)4	Mo	II	2.153	1.40	0	12	eclip	4	Cl	NR3	Cl	NR3	None
Mo2(CH2SiMe3)6	Mo	III	2.167	1.40	0	9	stag	3	CH2SiR3	CH2SiR3	CH2SiR3	None	None
Mo2(NMe2)4Cl2	Mo	III	2.200	1.40	0	9	stag	3	NMe2	NMe2	Cl	None	None
Mo2(NMe2)4Me2	Mo	III	2.201	1.40	0	9	stag	3	NMe2	NMe2	Me	None	None
Mo2(NMe2)6	Mo	III	2.211	1.40	0	9	stag	3	NMe2	NMe2	NMe2	None	None
Re2Cl6[PEt3]2	Re	III	2.222	1.37	0	12	eclip	4	PEt3	Cl	Cl	Cl	None
Mo2(OCH2CMe3)6	Mo	III	2.222	1.40	0	9	stag	3	OCR3	OCR3	OCR3	None	None
Re2Cl4[P(Et)3]4	Re	II	2.232	1.37	0	14	eclip	4	Cl	PEt3	Cl	PEt3	None
[Re2Cl8]2-	Re	III	2.241	1.37	-2	12	eclip	4	Cl	Cl	Cl	Cl	None
Mo2(OSiMe3)6HNMe2	Mo	III	2.242	1.40	0	11	stag	3	OSiR3	OSiR3	OSiR3	NR3	None
W2[CH2Si(CH3)3]6	W	III	2.255	1.41	0	9	stag	3	CH2SiR3	CH2SiR3	CH2SiR3	None	None
[Re2Br8]2-	Re	III	2.270	1.37	-2	12	eclip	4	Br	Br	Br	Br	None
W2(NMe2)4Cl2	W	III	2.283	1.41	0	9	stag	3	NMe2	NMe2	Cl	None	None
W2(NEt2)4Me2	W	III	2.291	1.41	0	9	stag	3	NEt2	NEt2	Me	None	None
W2(NMe2)6	W	III	2.294	1.41	0	9	stag	3	NMe2	NMe2	NMe2	None	None
W2(NEt2)4I2	W	III	2.296	1.41	0	9	stag	3	NEt2	NEt2	I	None	None
W2Cl2(NEt2)4	W	III	2.301	1.41	0	9	stag	3	Cl	NEt2	NEt2	None	None
W2(NEt2)4Br2	W	III	2.303	1.41	0	9	stag	3	NEt2	NEt2	Br	None	None
W2(OCHMe2)6(pyr)2	W	III	2.332	1.41	0	11	stag	3	OCR3	OCR3	OCR3	NR3	None
Co2(CO)6[P(n-Bu)3]2	Co	O	2.665	1.25	0	17	stag	1	CO	CO	CO	None	PEt3
Mn(CO)8[PEt3]2	Mn	O	2.913	1.37	0	17	stag	1	CO	CO	CO	CO	PEt3
Mn2(CO)10	Mn	O	2.923	1.37	0	17	stag	1	CO	CO	CO	CO	CO
Cr2(CO)10	Cr	-I	2.970	1.29	-2	17	stag	1	CO	CO	CO	CO	CO
Re2(CO)10	Re	O	3.020	1.37	0	17	stag	1	CO	CO	CO	CO	CO
Tc2(CO)10	Tc	O	3.036	1.35	0	17	stag	1	CO	CO	CO	CO	CO
Mo2(CO)10	Mo	-I	3.123	1.40	-2	17	stag	1	CO	CO	CO	CO	CO

REFERENCES

—, CRC Handbook of Chemistry and Physics, 65th Edition, R.C.Weast editor, CRC Press, Inc., 1984.

Aho,A.V., J.E.Hopcroft, J.D.Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

Becker,J.M., "AQ-PROLOG: A Prolog Implementation of an Attribute-Based Inductive Learning System," UIUCDCS-F-85-930, ISG 85-1, Department of Computer Science, University of Illinois, 1985a.

Becker,J.M., "Macros and Utilities for Franz Lisp," UIUCDCS-F-85-937, ISG 85-7, Department of Computer Science, University of Illinois, 1985b.

Becker,J.M., "Inductive Learning of Decision Rules with Exceptions: Methodology and Experimentation," University of Illinois, Department of Computer Science, M.S. Thesis, 1985c.

Bueche,F.J., Schaum's Outline of College Physics, Seventh Edition, McGraw-Hill Book Co., 1979.

Charniak,E, C.K.Riesbeck, D.V.McDermott, Artificial Intelligence Programming, Lawrence Erlbaum Associates, Publishers, 1980.

Condon,E.U., H.Odishaw (editors), Handbook of Physics, Second Edition, McGraw-Hill, 1967.

de Kleer,J., "Qualitative and Quantitative Knowledge in Classical Mechanics," M.S. Thesis, TR-352, Massachusetts Institute of Technology, Cambridge, MA, 1975.

Falkenhainer,B.C., "ABACUS: Adding Domain Constraints to Quantitative Scientific Discovery," UIUCDCS-F-84-927, ISG 84-7, Department of Computer Science, University of Illinois, 1984.

Falkenhainer,B.C., "Proportionality Graphs, Units Analysis, and Domain Constraints: Improving the Power and Efficiency of the Scientific Discovery Process," *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, 1985.

Forbus,K.D., "Qualitative Process Theory," PhD Thesis, TR-798, Massachusetts Institute of Technology, Cambridge, MA, 1984.

Huntley,H.E., Dimensional Analysis, MacDonald & Co., London, 1952.

Kokar,M., "A Procedure of Identification of Laws in Empirical Sciences", *Systems Science*, Vol. 7, No. 1, pp. 32-41, 1981.

Langhaar,H.L., Dimensional Analysis and Theory of Models, John Wiley & Sons, 1951.

Langley,P., "Rediscovering Physics with BACON.3," *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pp. 505-507, 1979.

Langley,P., "Data-Driven Discovery of Physical Laws," *Cognitive Science*, **5**, pp. 31-54, 1981a.

Langley,P., G.L.Bradshaw, H.A.Simon, "BACON:5 The Discovery of Conservation Laws," *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pp. 121-126, 1981b.

Langley,P., G.L.Bradshaw, H.A.Simon, "Rediscovering Chemistry with the Bacon System," In

R.S.Michalski, J.G.Carbonell, and T.M.Mitchell (editors), *Machine Learning - An Artificial Intelligence Approach*, Tioga Publishers, 1983a.

Langley,P., G.L.Bradshaw, H.A.Simon, J.Zytkow, "Mechanisms for Qualitative and Quantitative Discovery," *Proceedings of the International Machine Learning Workshop*, Monticello, Illinois, 1983b.

Langley,P., J.Zytkow, H.A.Simon, G.L.Bradshaw, "The Search for Regularity: Four Aspects of Scientific Discovery," to appear in *Machine Learning: An Artificial Intelligence Approach Vol. II*, R.S.Michalski, J.G.Carbonell, and T.M.Mitchell (Eds.), Morgan Kaufmann, 1985.

Lenat,D.B. "Automated Theory Formation in Mathematics," *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pp. 833-842, 1977.

Lenat,D.B. "The Role of Heuristics in Learning by Discovery: Three Case Studies," In R.S.Michalski, J.G.Carbonell, and T.M.Mitchell (editors), *Machine Learning - An Artificial Intelligence Approach*, Tioga Publishers, 1983.

Michalski,R.S., "Knowledge Acquisition through Conceptual Clustering: A Theoretical Framework and an Algorithm for Partitioning Data into Conjunctive Concepts", *Policy Analysis and Information Systems*, Vol. 4, No. 3, September 1980.

Michalski,R.S. "A Theory and Methodology of Inductive Learning," In R.S.Michalski, J.G.Carbonell, and T.M.Mitchell (Eds.), *Machine Learning - An Artificial Intelligence Approach*, Tioga Publishers, 1983.

Michalski,R.S. "Understanding the Nature of Learning: Issues and Research Directions," to appear in *Machine Learning: An Artificial Intelligence Approach Vol. II*, R.S.Michalski, J.G.Carbonell, and T.M.Mitchell (Eds.), Morgan Kaufmann, 1985.

Michalski,R.S., and J.B.Larson, "Selection of most representative training examples and incremental generation of VL1 hypotheses: The underlying methodology and the description of programs ESEL and AQ11," University of Illinois Technical Report UIUCDCS-R-78-867, May 1978.

Roller,D.E., and R.Blum, Physics: Volume One. Mechanics, Waves, and Thermodynamics, Holden-Day, San Francisco, 1981.

Stepp,R.E.III, "Conjunctive Conceptual Clustering: A Methodology and Experimentation," PhD Thesis, UIUCDCS-R-84-1189, Department of Computer Science, University of Illinois, 1984.

Winston,P.H. Artificial Intelligence, Second Edition. 1984 (p. 402-409).

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-85-947	2.	3. Recipient's Accession No.

4. Title and Subtitle

Quantitative Empirical Learning: An Analysis and Methodology

5. Report Date: August 1985

6.

7. Author(s)

Brian C. Falkenhainer

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois
Urbana, IL 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.
NSF DCR 84-06801
ONR N00014-82-K-0186

12. Sponsoring Organization Name and Address

National Science Foundation, Washington, DC
Office of Naval Research, Arlington, VA

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

This thesis discusses the issues involved in quantitative empirical learning and presents a theory of quantitative learning which is embodied in a program called ABACUS. ABACUS formulates equations that cover subsets of observed numerical data and derives explicit, logical descriptions, called domain constraints, which state when each equation is applicable.

Several new techniques particularly suited to quantitative learning are introduced in this work. Units Analysis enables one to greatly reduce the size of the search space by examining the compatibility of variables' units. Two new search algorithms, proportionality graph search and suspension search address some of the unique search problems associated with quantitative learning. Finally, a number of examples are presented and analyzed which demonstrate the capabilities of the ABACUS quantitative learning system.

17. Key Words and Document Analysis. 17a. Descriptors

Machine Learning
Quantitative Learning
Inductive Learning
Learning by Discovery

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 107
	20. Security Class (This Page) UNCLASSIFIED	22. Price