J. M Bishop
24/3/91

South African Computer Journal
Number 4
March 1991

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 4
Maart 1991

# Computer Science and Information Systems

# Rekenaarwetenskap en Inligtingstelsels

**The South African Computer Journal**
*An official publication of the South African Computer Society and the South African Institute of Computer Scientists*

**Die Suid-Afrikaanse Rekenaartydskrif**
*'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes*

**Editor**
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083

**Assistant Editor: Information Systems**
Dr Peter Lay
P. O. Box 2142
Windmeul 7630



**Subscriptions**

| | Annual | Single copy |
|---|---|---|
| Southern Africa: | R32-00 | R8-00 |
| Elsewhere: | $32-00 | $8-00 |

to be sent to:
*Computer Society of South Africa*
*Box 1714 Halfway House 1685*

Computer Science Department, University of Pretoria, Pretoria 0002
Phone: (012) 420-2843 Fax: (int) + (012) + 43-6454

Sigma Press

# Guest Editorial

# Does Today's Industry Need Qualified Computer Scientists?

This guest editorial consists of two contrasting views on the value to industry of a professional degree in computer science. Both authors, one local and one from Germany, are managing directors of well-respected software houses. (Editor)

## Viewpoint I

Hans G Steiner
*MBP Software and Systems GMBH*
*Semerteichestrasse 47-49*
*D4600 Dortmund 1*

I would like to begin by recounting from my student days a story that I consider to be relevant. While attending a career forum for computer scientists, mathematicians and physicists, the personnel officer from IBM Germany was asked if he would consider taking on mathematicians. The gist of his answer was as follows: "Of course I must admit that I could just as well give the mathematician's job to a theologian. What is important is the ability to think logically. It is only there, on the job, that he learns how to become productive for us."

This episode occurred 14 years ago at a time when graduating mathematicians did not necessarily learn programming and when computer scientists were few and far between. The situation has improved immensely since then. Mechanical engineers, electrical engineers and physicists, all with programming knowledge, have for the most part taken over many programming jobs. This shows industry that, as time goes by, the answer to the opening question is becoming an ever-louder and more frequent "NO".

I support this opinion and in the remainder of this essay I will expand on my reasons, as well as highlight some exceptions.

An employee who is recruited directly from a university should possess the following four capabilities:
*1. An ability to think logically:* One of the basic requirements in our business is the ability to recognise, analyze, structure, break down and solve a problem as well as to fully synthesize the solution. The important thing is to break down the problem in such a way that the individual components can feasibly be solved. This is what distinguishes an engineer/scientist from an arts scholar. The latter usually concentrates on the complete problem and tends to settle for a contentious, complex and partially non-feasible solution. In our business, it is not enough to merely ask the "right" questions.

This ability to think logically may be accentuated in computer science; however engineers/scientists will generally possess the ability to an equal extent.

*2. Programming skills:* Our employees' prime tool of the trade is their ability to encode solutions to problems. Ideally, this ability ought to be held as abstract as possible. In other words, the further away from the "bit", the better. FORTRAN programmers who, for example, concentrate on the multiple use of memory space of all variables will never be successful programmers in an object-oriented programming language.

The difference can be seen, even in today's universities. For example, one only has to read a PROLOG program from a student who learned PASCAL in his first semester and PROLOG in his fifth. On average, this is always a "PASCAL program in PROLOG". The various possibilities offered by a predicate calculus language are only recognised and used by the best students. Again, we do not need the average computer science scholar who has spent between six and eight years writing complicated PASCAL programs, but rather the "thinker" with basic programming knowledge who is capable of abstracting the task. Once again, the ability is independent of faculty.

*3. Teamwork skills:* Working successfully in a team

This SACJ issue is sponsored by
**Department of Computer Science**
**Rhodes University**

requires assertiveness, tolerance, stability and one's own ideas. Very few problems have solutions that can be managed by one person successfully in the allocated time. Out of 700 employees, we can only afford approximately five "lone warriors" who are, in turn, the leading specialists in a wide field. They have a strategic vision which we follow. All remaining employees are evaluated, for better or worse, on their team performance. Some people have an in-built ability to work in teams. A few universities - unfortunately not enough - encourage this team-thinking. Again we see that the ability is independent of university faculty.

4. *Motivation:* The ability to enjoy one's particular job is a major driving force in every employee. Whereas in the sixties everything had to be "bigger, faster and better" and in the eighties "things had to be meaningful to society", the theme for the nineties is self-realization. Those companies who succeed in incorporating different employees (ie employees with different driving forces) into the company culture and who motivate each employee optimally will be successful in the nineties and beyond. There are huge productivity gains to be had from motivating employees. Compared with this, the possibilities offered by CASE tools pale into insignificance.

One basic requirement is thus the recruitment of a self-motivated employee who should at no stage become demotivated, whether it be by company culture, superiors or working conditions.

Again, this is not linked to a specific university faculty and is independent of know-how.

As none of these four capabilities are necessarily restricted to studies in computer science, the technical/-scientific background of new employees who are being recruited is largely irrelevant.

I would now like to point out a few exceptions which might give a computer scientist the upper hand in an interview. I refer exclusively to our own company and our specific company tasks.

1. Porting our COBOL Compiler onto the latest UNIX machine from the manufacturer XY. Knowledge of the UNIX operating systems could be very valuable and enable the new employee to rapidly become productive.
2. Programming the 37th interface (special customer request) for our ISDN card. Knowledge of interface protocols or experience with protocol conversions would be very useful and could be a decisive factor. Such specialized knowledge is usually very rare.
3. Adapting our integrated office automation system to the 17th foreign language. The employee must command the language perfectly. Simply outsourcing the translation would mean that this language version could not be maintained or supported. From this example one can see that specialized knowledge not only refers to knowledge gained from computer science studies.

In the product business, it sometimes happens that computer scientists with specialized knowledge are sought. (This is almost impossible in the project business, due to the variety of tasks to be performed.) However such a "knowledge" advantage over others usually only lasts about a year. After that, the achievements of two different employees (one with specialized knowledge and the other without) tends to even out.

Most applicants who start out do not know our products, as the flow of employees in this industry is almost always from manufacturer to user. Hardware and software manufacturers often lose their products specialist to the products' users. Seldom do employees change in the other direction.

In my opinion, universities can learn two things from this essay:

1. Studies in computer science give basic knowledge that can be used in various jobs. The student should however be careful not to place all his eggs in one basket.
2. Teamwork should be encouraged more. Time allows for very few geniuses, acting as 'lone warriors', to initiate progress in our society.

I have taken the liberty of basing my interpretation and answer to the opening question on my own judgement and experiences. I would be grateful for other opinions and experiences on this topic.

I would like to conclude by expressing my gratitude for having had this opportunity to express my views.

---

## Viewpoint II

Pierre Visser
*Grinaker Informatics, P.O.Box 29818, Sunnyside, 0132*

The title question currently generates as many viewpoints as a counterpart question: "What is the correct curriculum for a computer science qualification?" Such questions stem from the many and diverse requirements expected to be fulfilled by the still developing applied science. A basic assumption of this editorial is that we need to have an explosion and consolidation in computer science theory. Only after this has occurred will a more general consensus of opinion exist - as is the case in other matured sciences.

An argument is presented here for the current approach of striving towards a balance between immediate industry needs and long term perceived theoretical requirements of industry, even though the balance, as viewed from either side, will always be imperfect.

Industry can, of course, do without qualified computer scientists - that is how it was established. Dedicated mathematicians, physicists, engineers and other scientists will, as in the past, continue to effect improvements. However, as one of those scientists from the

early days, it is difficult for me to understand why one would choose to continue this way.

A computer science qualification is viewed here as a university education (4 years) into theory that is not obtainable otherwise. By definition, therefore, a qualified computer scientist is not trained to conform to specific job requirements. Rather, the computer scientist will possess knowledge that will serve him long past the present day's computing technology.

Whether industry needs qualified computer scientists depends on two issues. Firstly, can an education be provided for computing technology that will serve as a foundation for the student's next 45 years in industry; and secondly, can industry build upon this foundation to created wealth more effectively than without qualified computer scientists.

It is widely accepted that, in broad terms, the teaching of fundamental theory will serve the first purpose. However, what subject matter to include from the wealth of mathematics, physics, OR, and from computing fields such as networking, operating systems and others, remains the illusive issue. Universities can merely strive to select the right mix for the perceived future needs of industry. This requires insight into the evolution of computing technology. I will later discuss such insight as a basic requirement for a qualified computer scientist.

What is important in teaching is to focus on fundamental theory. Just as the natural science student needs to breed fruit flies in order to gain insight into the dynamics of inheritance, so too the computer science student needs to develop software. The purpose should be to create understanding and insight into fundamental theory, and, just as in the case of the breeder of fruit flies, the software developed should never be measured against efficiency requirements from industry.

The second issue is whether industry can build on this theoretical foundation to create wealth.

A depth of insight into computing technology, more so than with other training, can be identified as the focus of the potential value of a qualified computer scientist to industry. Three areas which require such insight are discussed below, namely organisation, product definition and the application of new computing technology in industry.

Computing products form an integral part of an organisation, and represent a significant capital investment aimed at increasing efficiency. These products are incorporated in an evolutionary way to match changing organisational requirements with improving product capabilities. Decisions to use products determine the long term efficiency and cost-effective replacement. Such decisions require insight into computing technology and its evolution. A qualified computer scientist can improve such decisions only if he gains enough insight into computing technology as well as its interaction with business through years of practice.

The success of products in some areas is dependent on market requirements which depend on computing technology and its evolution. The correct definition of characteristics of products that interface to computers is such an example. Insight into computing technology is able to create the versatility, simplicity or other improved selling features which can open new market segments.

The third area where insight into computing technology plays an important role is in the application of new computing technology (or a new trend) in an organisation. Examples include the introductory period for networking, DBMS-technology, distributed processing and document image processing. In areas such as these, the newly qualified computer scientist can be applied effectively and at the same time build up insight through experience which he will require for the other areas of organisational and product decisions mentioned above.

A major dilemma in the continuous development of insight into computing technology by qualified computer scientists is their correct application in industry. The identification of the opportunities within the three areas discussed above, requires insight into computing technology itself. Winning companies that depend on computing technology have this ability. In such companies the insight of the qualified computer scientist into computing technology as well as its contribution to the business is constantly stimulated, turning the qualified computer scientist into a valuable company resource.

What has been neglected in this whole discussion is the role of the "technician" and of the casual user of computing technology. Such personnel are required to implement selected computing technology of the day efficiently, whether in accounting, chemical engineering or other specialised disciplines. Their role and place is unquestioned. However, it cannot be expected of them to evaluate the potential of new computing technology, formulate algorithms from fundamental theory or any such decisions which require insight built upon a sound theoretical knowledge of the field.

The final aspect in answering the opening question is whether the qualified computer scientist can outperform other professionals who build up their own experience in computing technology. Many examples could be cited of improvement brought about by non-computer scientists in the past. However, these individuals formed part of the bootstrapping for computer science theory and education. We should have faith in this bootstrapping of computer science qualifications, because computing technology will increasingly diversify into many directions of specialisation in years to come, each requiring a body of fundamental theory.

This complexity cannot be left to a casual development of insight - industry requires qualified computer scientists to experience interaction with business objectives in order to cope successfully with future computing technology.

# CID3: An Extension of ID3 for Attributes with Ordered Domains

I Cloete and H Theron
*Department of Computer Science, University of Stellenbosch, Stellenbosch, 7600*

## Abstract

*Quinlan's ID3 is a popular and efficient algorithm for inducing decision trees from concept examples, where the examples are presented as vectors of attribute-value pairs. If some attributes have integer or real domains ID3 tends to generate very complex decision trees. This is due to: (1) an attribute selection heuristic biased towards attributes with domains of large cardinality (2) strong constraints (bias) imposed on decision trees generated and (3) the fact that ID3 does not distinguish between attributes with unordered domains and attributes with linearly ordered (integer or real) domains. ID3-IV and GID3 address the first and second problem respectively. We propose CID3, a generalization of GID3, which addresses the third problem. These algorithms are compared with respect to five criteria for decision tree quality and computational efficiency. The test domain consists of normal and abnormal electrocardiograms (ECGs) described mainly by integer and real attributes. CID3, which implements the weakest bias and uses the most domain knowledge, generates a superior quality decision tree for the ECGs.*
***Keywords:*** *Machine learning, Induction, Decision trees*
***Computing Review Categories:*** *I.2.5, I.2.6*

Recieved July 1990, Accepted November 1990

## 1. Introduction

Concept learning (also referred to as concept acquisition or learning from examples) is an area of machine learning which has received considerable attention. Given a set of concepts and examples of each concept, a concept learning program attempts to induce a description that distinguishes among the concepts. Quinlan's ID3 [4], a descendant of Hunt's CLS [3], is an example of a concept learning program which induces such a description in the form of a decision tree (Figure 1b). ID3 may be characterized as follows:

Given:

- A set of concepts.
- A set of attributes and the domain of each attribute.
- Examples of each concept, where an example consists of a vector of attribute-value pairs.

It will induce:

- A decision tree which distinguishes among the concepts.

ID3 regards the domain of every attribute as a set of discrete values (nominal unordered values such as `sex` $\in$ {`male`, `female`}). If some attributes have integer or real domains (i.e. domains with large cardinality) ID3 tends to generate very wide and shallow decision trees. These trees contain many classification rules (each path from the root to a leaf corresponds to a classification rule). Such a large number of rules is difficult to understand and interpret. A further drawback is that these trees fail to generalize over the examples due to insufficient support for each rule.

The low quality trees generated by ID3 when integer or real attributes are present are due to:

- An attribute selection heuristic biased towards attributes with domains of large cardinality:

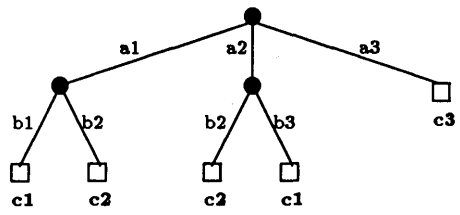
  ID3 regards the domain of every attribute as a set of unordered values. An attribute such as the amplitude of a continuous analog signal can be represented by sampled finite real numbers within an interval. In that case most examples assume different values in a given training set. ID3 views such attributes simply as unordered values over a domain with large cardinality. When ID3 expands a node in a decision tree, it employs an information theoretic heuristic (section 2.1) to pick the best attribute (*split attribute*) on which a node is split. This heuristic favours domains with large cardinality. Thus ID3 will tend to select attributes with integer or real domains as split attributes. Since ID3 creates a branch for each value of a split attribute which occurs in the training examples (it creates a *complete partition* of a split attribute's valueset), this bias results in decision trees with many branches emanating from each node. This in turn leads to decision trees which contain many leaves (classification rules). ID3-IV [2] addresses this problem by implementing a heuristic which is not biased towards large valuesets.

Attributes: A ∈ {a1, a2, a3}
B ∈ {b1, b2, b3}

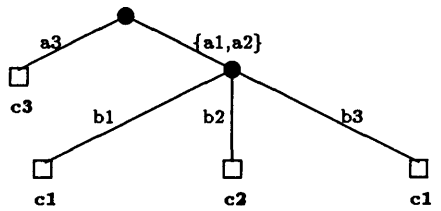
Concepts: {c1, c2, c3}

| | A | B | Concept |
|---|---|---|---|
| e1 | a1 | b1 | c1 |
| e2 | a1 | b2 | c2 |
| e3 | a2 | b2 | c2 |
| e4 | a2 | b3 | c1 |
| e5 | a3 | b1 | c3 |
| e6 | a3 | b2 | c3 |

(a) Example of input to a decision tree program.




(b) Decision tree generated by ID3.




(c) Decision tree generated by GID3 with TL=1.0

**Figure 1: Examples of decision trees [2].**

- Strong constraints (bias) imposed on decision trees:

  As stated above, ID3 creates a complete partition of a split attribute's valueset. This built-in bias excludes more general attribute partitions where branches may be labeled with subsets of a split attribute's valueset, which in turn may lead to simpler decision trees. GID3 [2], a generalization of ID3-IV, addresses this problem by partitioning an attribute's valueset such that all irrelevant attribute values are grouped together (Figure 1c).

- ID3 assumes that all attributes have unordered domains:

  As stated above, most examples usually take different values for integer or real-valued attributes, causing ID3 and ID3-IV to create many branches when splitting a node on such an attribute. GID3 partly solves this problem by creating one branch for all irrelevant values. Despite this generalization, the following drawbacks remain: (1) An integer or real attribute's entire domain is not covered by the partitions created by these algorithms, and (2) branches are labeled with single values or a set of values (GID3) rather than with relational tests. CID3 was designed to solve these problems. CID3's background knowledge includes a distinction between attributes with unordered domains and attributes with ordered (i.e. integer or real) domains. It splits an ordered attribute's valueset into the two intervals $(A \leq a_i)$ and $(A > a_i)$ completely covering its domain. These intervals are more general and concise than those generated by the other algorithms.

This paper empirically compares the ID3, ID3-IV, GID3 and CID3 algorithms. The test domain consists of examples of electrocardiograms (ECGs). The standard ECG consists of 12 leads which are labeled V1 to V6, I to III, aVL, aVR and aVF. These leads are placed on a patient's chest, arms and legs. Each lead measures electrical conductivity in the heart from a different angle. Every heart beat generates a series of positive and negative deflections in the ECG wave. These deflections are called the P, Q, R, S, and T waves. Diagnoses of ECGs are made based on the morphology of these waves. Each ECG was described by 201 attributes. Of these attributes, 198 had real domains, two (`atrial` and `ventricular rate`) had integer domains, and one (`sex`) had an unordered domain. Examples of the real attributes are `V1-P-amplitude` (P-wave amplitude in lead V1) and `V6-Q-duration` (Q-wave duration in lead V6). The aim of the experiments was to induce a decision tree which distinguishes accurately between normal and abnormal ECGs. This problem was selected because the distinction between normal and abnormal ECGs is the most difficult aspect of ECG diagnoses due to great variations in ECG appearance within a population of normal subjects, and also the wide range of different appearances amongst the 12 leads of a single normal subject [5].

The remainder of this paper is structured as follows. Section 2 describes the four different algorithms. Section 3 provides a comparison of these algorithms. The algorithms are compared with respect to computational and classification efficiency and four criteria for decision tree quality. The experiments show that CID3 generates the simplest, most efficient and most accurate decision trees.

## 2. Four Algorithms for Tree Induction

Algorithm 1 outlines the general structure of ID3, ID3-IV, GID3 and CID3. These algorithms differ in the way that they implement steps (1) and (2). In the following sections it is shown how each algorithm implements steps (1) and (2), and how it generalizes

Suppose a node consists of three fields:

- attr: The attribute used to split the tree at the node.
- concept: Undefined or the name of one of the concepts.
- children: A list of records with fields: testset, subtreeptr where testset is the value / subset of attr that is the condition to the branch, and subtreeptr points to the subtree for that value / subset.

```
proc BuildTree (AttributesAndValueSets,examples)
begin
  create a node {the root} with:
    class := undefined;
    attr := undefined;
    children := nil;
  ExtendTree(AttributesAndValueSets,examples,node)
end; {BuildTree}

proc ExtendTree (AttributesAndValueSets,examples,node);
begin
  if (examples all in same class) then
    node.class := class to which all the examples belong
  else begin
    partitions := PartitionAttr(AttributesAndValueSets,
                                examples);          (1)
    node.attr := FindBestSplitAttr(partitions,examples); (2)
    for each subset Vi in the partition of node.attr do:
      Create newnode with:
        class := undefined;
        attr := undefined;
        children := nil
      Add newnode to the children of node by creating a new
      child record with:
        testset := Vi;
        subtreeptr := newnode
      ExtendTree(AttributesAndValueSets -
                 {node.attr and its valueset},
                 examples which take a value in Vi
                 for node.attr, newnode)
    end {for}
  end {else}
end; {ExtendTree}
```

**Algorithm 1: A general tree induction algorithm.**

its predecessor. The experimental results obtained with these algorithms for the ECG experiments are described in section 3.

### A Reconstruction of ID3

Quinlan's ID3 [4] implements strong bias with respect to the choice of a split attribute and the partitioning of its values. It *restricts the decision trees* generated to allow only complete partitions (step (1) in algorithm 1). Thus whenever a node is split on an attribute, a new child is created for each value of that attribute. ID3 employs an information theoretic *heuristic* to pick the best split attribute (step (2) in algorithm 1). The aim of this approach is to find a decision tree having the minimum number of tests to classify an example. Let $S$ be a set of examples to be classified at a node $N$ in a decision tree. Let $p(C_i)$, $C_i \in Concepts$, denote the proportion of examples in $S$ which belong to class $C_i$. The *information content* of $S$ is given by:

$$I(S) = - \sum_{C_i \in Concepts} p(C_i) \log_2 p(C_i)$$

Now suppose that all the examples in $S$ do not belong to the same class. The best attribute on which to split $N$ is the one which leads to the largest gain in information where

*Gain = information before split - information after split.*

Information before the split is given by $I(S)$. Assume that $N$ is split on attribute $A$ which assumes values $V_i \in valueset(A)$, where $valueset(A)$ are all those values of $A$ occurring in the examples in $S$. The information content of $S_i$ is given by $I(S_i)$ where $S_i$ is the subset of examples in $S$ with value $V_i$ for attribute $A$. The information content after the split (the *entropy* of $A$) is defined as the weighted average of the information content of each subset $S_i$ of $S$:

$$E(A,S) = \sum_{V_i \in valueset(A)} \frac{|S_i|}{|S|} * I(S_i)$$

Thus the best attribute to split on maximizes:

$$Gain(A,S) = I(S) - E(A,S).$$

For some integer and real-valued attributes in the ECG classification problem, nearly all the examples assume a different value. Since ID3 does not distinguish between ordered and unordered attributes, it assumes these attributes are unordered and have domains with large cardinality. Viewed this way, these attributes carry the most information and is favoured by the *gain* heuristic. As a result very wide and shallow decision trees are generated where most examples may be classified by testing only one or two attributes. This type of decision tree is complex and fails to capture the underlying structure of the ECG domain.

### ID3-IV: Using Another Heuristic

In order to compensate for domains with large cardinality, Quinlan [2] modified the gain heuristic as follows:

$$unbiased\text{-}gain(A,S) = \frac{gain(A,S)}{compensation(A)}$$

with

$$compensation(A) = \sum_{V_i \in valueset(A)} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

where $S$ is the set of examples to be classified at a node and $S_i$ is the subset of $S$ which takes value $V_i$ for attribute $A$.

The only difference between ID3 and ID3-IV is that the latter uses *unbiased-gain(A,S)* instead of $gain(A,S)$ for step (2) in algorithm 1. Employing this heuristic usually results in slightly better decision trees. However, for the ECG classification problem splitting on attributes with domains of large cardinality is only deferred to lower branches in the decision

tree. Thus, although slightly deeper decision trees are generated by ID3-IV, they still have the same drawbacks as the trees generated by ID3.

### GID3: Relaxing Constraints on Trees

The main shortcoming of the decision trees generated by ID3 and ID3-IV is that they fail to generate trees which generalize sufficiently over the examples. This problem may be solved by generating more general (than complete) partitions for a split attribute's valueset, where a *partition* $P$ of a set $A$ is a set of sets (called *blocks*) $\{A_1, A_2, \ldots, A_n\}$ such that $A_i \cap A_j = \emptyset$ for all $i \neq j$ and $\bigcup_i A_i = A$. The number of ways to partition an $n$-element set into $k$ blocks is given by *Stirling's formula of the second kind* [1]:

$$\begin{aligned} S(n,k) &= S(n-1,k-1) + kS(n-1,k), \\ &\quad \text{where } 1 \leq k \leq n \\ S(0,0) &= S(n,n) = S(n,1) = 1 \end{aligned}$$

For example, $S(4,3) = 6$, $S(10,2) = 511$, $S(10,5) = 42525$ and $S(20,5) > 7 \times 10^7$. It is clear that even for attributes with relatively small valuesets, it is not feasible to generate all possible partitions and then pick the "best" one. The GID3 and CID3 algorithms described below attempt to find a "good" partition without investigating all possible partitions.

GID3 [2] partitions an attribute's valueset by placing all irrelevant values into one block. This results in a tree where one branch is labeled by a block of irrelevant values and each of the remaining branches emanating from the same node is labeled by one of the relevant values. Irrelevant attribute values are determined as follows (algorithm 2): For each value $V_i$ of each attribute $A$, the partition $\{\{V_i\}, valueset(A) - \{V_i\}\}$ is created and the entropy of this partition is calculated. This amounts to determining the classification information of each value $V_i$. The lowest entropy of all these partitions is denoted by *Min-E*. This value is used to determine an entropy threshold: *threshold* = *Min-E* * *TL*, where *TL* (tolerance level) is a user specified value. All those values whose corresponding partition has entropy higher than *threshold* are irrelevant. The irrelevant values are grouped together in one block. If all the values of an attribute are irrelevant, this attribute may be ignored (this is the purpose of *Min-E-$A_i$* in algorithm 2). In this way a partition for each attribute (except those which are ignored) is created.

From the description of GID3 it can be seen that the purpose of *TL* is to determine which values are to be deemed irrelevant. Using $TL = 1.0$ will usually result in a binary tree since only those values whose corresponding partition has entropy exactly the same as *Min-E* will not be deemed irrelevant. On the other extreme, $TL = \infty$ will produce complete partitions, resulting in the same tree that ID3 would generate. Ideally, a *TL* value which will result in the simplest decision tree with the highest classification accuracy is sought. Cheng et al [2] claims the existence of a value for *TL* which will result in a better quality decision tree than that of ID3, but states that predicting such a value for *TL* remains as future work. Thus, for the time being, an appropriate value for *TL* can only be found through experimentation. For an example of a tree generated by GID3 with TL = 1.0 see Figure 1c.

Replace step (1) in algorithm 1 with the following:

1. For each attribute $A_i$ let *valueset*($A_i$) be the set of its values occurring in the examples at the node to be split.
2. Min-E := $\infty$;
3. For each attribute $A_i$ with | *valueset*($A_i$) | > 1 do:
   {*The test above ensures that $A_i$ carries useful information.*}
   Min-E-$A_i$ := $\infty$;
   For each value $V_i$ in *valueset*($A_i$) do:
   partition := $\{\{V_i\}$, *valueset*($A_i$) - $\{V_i\}\}$;
   Determine *entropy*($S, A_i$) where the
   domain of $A_i$ = partition;
   If *entropy*($S, A_i$) < Min-E then
   Min-E := *entropy*($S, A_i$);
   If *entropy*($S, A_i$) < Min-E-$A_i$ then
   Min-E-$A_i$ := *entropy*($S, A_i$);
4. Threshold-E := Min-E * TL
   {*TL (tolerance level) is a user specified value*}
5. For each attribute $A_i$ with Min-E-$A_i$ < Threshold-E do
   *newvalueset*($A_i$) := $\emptyset$;
   default := $\emptyset$;
   For each partition $\{\{V_i\}$,*valueset*($A_i$) - $\{V_i\}\}$ of $A_i$
   created in step 3 do:
   if the *entropy*($A_i, S$) corresponding to this
   partition < threshold-E then
   *newvalueset*($A_i$) := *newvaluesset*($A_i$) ∪ $\{V_i\}$
   else
   default := default ∪ $\{V_i\}$;
   if default <> $\emptyset$ then
   *newvalueset*($A_i$) := *newvalueset*($A_i$) ∪ {default}
   Let *newvalueset*($A_i$) be the partition of $A_i$.
6. Execute the rest of ID3-IV.

**Algorithm 2: A reconstruction of GID3.**

The only difference between ID3-IV and GID3 is that GID3 uses algorithm 2 to partition an attribute's valueset while ID3-IV creates a complete partition. In section 3 it will be shown that for TL = 1.0, this relatively small difference results in a much more compact decision tree for the ECG classification problem.

### CID3: Adding More Domain Knowledge

GID3 creates attribute partitions consisting of single element sets and possibly one set with more than one element. Thus for a real attribute like **amplitude** with domain 0..100, GID3 may form a partition such as {{1}, {4}, {10}, {30}, {2,6,7}}. As stated in the introduction, this type of partition does not entirely cover an integer or real attribute's domain. If a decision tree with a branch labeled by this partition should be used to classify new (test) examples, it cannot classify those examples which take values for **amplitude** that does not occur in the decision tree. Another drawback of this type of partition is that it is difficult to understand since it does not highlight those features which distinguish among the concepts. For many problems, examples which belong to the same class usually take similar values for integer and real attributes, e.g. "ex-

ample E belongs to concept C if its value for amplitude is below 40". This type of conditions cannot be generated by the algorithms described above.

To solve these problems, we developed CID3. The domain knowledge of CID3 distinguishes between attributes with ordered and unordered domains. For unordered attributes, GID3's attribute partitioning mechanism (algorithm 2) is used. For an ordered attribute $A$, algorithm 3 is employed to partition its valueset: First, all the values of $A$ are sorted from small to large. Then a partition consisting of two intervals $(A \leq a_i)$ and $(A > a_i)$ is created for each value $a_i$ except the largest value (otherwise the interval $(A > a_i)$ is empty). The partition with the lowest entropy is kept as the best partition for $A$. An advantage of this type of partition above those partitions generated by GID3 is that they completely cover the domain of an ordered attribute.

CID3 may be generalized to generate partitions consisting of more than two blocks. Algorithm 4 describes a straight forward extension of CID3 which generates partitions consisting of three blocks. The following analysis shows that this algorithm is too inefficient to be practical for partitioning ordered attributes which take many different values in a given training set. Let $A$ be an ordered attribute with $n$ values. The number of partitions investigated by CID3 (algorithm 3) are (n - 1). Thus the time complexity of CID3 is $O(n)$. On the other hand algorithm 4 has to investigate $(n-2)(n-1)/2$ partitions. Thus this algorithm's time complexity is $O(n^2)$. If $n$ is 500 then 124251 partitions must be investigated. The same holds for extensions of algorithm 4 which generate partitions consisting of more than three blocks. To aggravate the situation even more, all two-, three-, four-, ..., n-partitions must be created and compared to find the best partition of all. Due to their inefficiency, no algorithms which generate partitions consisting of more than two blocks were implemented.

The decision trees generated by CID3 for the ECG classification problem were slightly deeper than those generated by the other algorithms. This is because it partitions an ordered attribute's values into only two intervals. However, the decision trees generated by CID3 have much greater predictive power than the trees generated by the other algorithms (see next section). In addition, these trees are easier to understand.

1. Assume that $A$ is an ordered attribute which takes more than one value in the examples at a node. Let *valueset*($A$) be the set of values the examples at that node takes for $A$.
2. Sort *valueset*($A$) from small to large.
3. For each value $a_i$ in *valueset*($A$), except the largest value, determine the *entropy* of $A$ with the partition $\{(A \leq a_i), (A > a_i)\}$.
4. Keep the best such partition as the partition for $A$.

**Algorithm 3: CID3 for partitioning an ordered attribute's valueset.**

1. Assume that $A$ is an ordered attribute and let *valueset*($A$) be the set of values the examples at that node takes for $A$.
2. Sort *valueset*($A$) from small to large.
3. For each value $a_i$ in *valueset*($A$), except the largest two values do:

   For each value $a_j$ in *valueset*($A$), $a_j > a_i$, except the largest value, determine the *entropy* of $A$ with the partition $\{(A \leq a_i), (a_i < A \leq a_j), (A > a_j)\}$.
4. Keep the partition with the lowest entropy as the new valueset of $A$.

**Algorithm 4: CID3 generalized to generate partitions with three blocks.**

## 3. Empirical Comparison

The previous section described four variants of ID3. In this section these algorithms are compared using ECG data. In total 846 ECGs were available. Of these 213 were normal and 613 abnormal. Each experiment was repeated 5 times and the average results are reported. For each iteration, 70% of the ECGs were randomly selected as a training set and the remainder was used for testing the classification accuracy of the induced trees. All the experiments were performed on a VAX series 6000 model 410 using COMMON LISP.

The algorithms were compared using the following criteria (based on those provided in [2]):

1. CPU time needed to generate a decision tree: This is a measure of the computational efficiency of the algorithm. The remaining measures are criteria for comparing decision tree quality.
2. The CPU time required to classify a set of test examples. This is an indirect measure of the complexity of an induced decision tree since it gives an indication of the number of tests that were required to classify the test examples.
3. The number of nodes in a tree: This is a measure of the complexity of the decision tree.
4. The number of rules (leaves) in a tree: The path to each leaf corresponds to a classification rule of the form "if (A $\in$ {a1,a2}) & (B $\in$ {b1,b3}) & ... & (K $\in$ {k4}) then the example belongs to Concept$_i$" where A, B and K are attribute names. Trees with fewer rules are preferred since they are easier to understand.
5. Average example support per rule: This is the average number of examples in the training set

classified by each leaf (rule) in the decision tree. It is a measure of the applicability (generality) of each rule.

6. The predictive power of the decision tree: We distinguish among: (1) The number of test examples correctly classified, (2) the number of test examples incorrectly classified, and (3) the number of test examples which could not be classified.

For the ECG domain all the attributes, excluding the attribute **sex**, had integer or real domains. Since ID3, ID3-IV and GID3 label branches with single values or sets of values it should be expected that all test examples cannot be classified. In order to classify all test examples, the following strategy was followed[1]: For trees generated by ID3 or ID3-IV an example is classified down the branch whose value equals or is closest to the value taken by that example. If more than one branch qualify, the first one found is selected. For trees generated by GID3, the matching problem is circumvented by classifying an example down the branch labeled by the subset of irrelevant values if it does not match one of the branches labeled by a single value.

Tables 1 and 2 summarize the results obtained with each of the algorithms. ID3 generated complex and inaccurate decision trees, even when closest match classification was performed. ID3-IV performed similar to ID3 with respect to most criteria. Thus employing an unbiased attribute selection heuristic did not lead to significantly simpler or more accurate decision trees. For GID3 trees were generated for TL values of 1.0, 1.1, 1.2, 1.5 and 2.0. The simplest and most accurate trees (when closest match classification was performed) were generated for a TL value of 1.0. These trees contained only 58.6 rules on average compared to the 507.0 and 479.4 rules for trees generated by ID3 and ID3-IV. Thus creating more general partitions where some branches are labeled with subsets of values did result in better quality trees. The price for this improvement was a large increase in the time required to induce the trees. TL values larger than 1.0 resulted in trees similar in quality to those generated by ID3 and ID3-IV. CID3 generated the best quality trees with respect to all criteria. These trees contained only 33 rules on average and correctly classified 85% of the test examples. The main reason for CID3's success is that it labeled branches with intervals which completely cover the domains of attributes with integer and real domains. Although CID3 consumed more time than ID3 and ID3-IV, it required considerable less time to induce a tree than GID3 with TL = 1.0. This is explained by the fact that GID3

[1] This classification strategy was suggested by an anonymous referee.

| | GT | #N | #R | ARS |
|---|---|---|---|---|
| ID3 | 394.7 | 554.0 | 507.0 | 1.17 |
| ID3-IV | 442.3 | 524.2 | 479.4 | 1.24 |
| GID3 TL=1.0 | 4445.4 | 114.2 | 58.6 | 10.11 |
| GID3 TL=1.1 | 938.2 | 403.6 | 368.7 | 1.61 |
| GID3 TL=1.2 | 597.8 | 515.8 | 470.0 | 1.26 |
| GID3 TL=1.5 | 623.3 | 515.8 | 470.0 | 1.26 |
| GID3 TL=2.0 | 579.3 | 516.4 | 471.6 | 1.26 |
| CID3 | 3097.6 | 65.0 | 33.0 | 18.0 |

Legend

GT: Tree generation time (seconds).
#N: The number of nodes in a tree.
#R: The number of rules (leaves) in a tree.
ARS: The average example support per rule.

**Table 1: Complexity of induced decision trees.**

| | CT | %C | %W | %UC |
|---|---|---|---|---|
| ID3 | 3.80 | 29.1 | 12.7 | 58.2 |
| ID3 & closest match | 16.52 | 65.1 | 34.9 | 0 |
| ID3-IV | 3.82 | 30.00 | 14.1 | 55.9 |
| ID3-IV & closest match | 13.30 | 63.9 | 36.1 | 0 |
| GID3 TL=1.0 | 0.52 | 18.3 | 7.8 | 73.9 |
| GID3 TL=1.0 & closest match | 0.58 | 72.7 | 27.3 | 0 |
| GID3 TL=1.1 | 2.61 | 24.6 | 10.0 | 65.4 |
| GID3 TL=1.1 & closest match | 8.89 | 68.9 | 31.1 | 0 |
| GID3 TL=1.2 | 3.54 | 28.4 | 12.8 | 58.8 |
| GID3 TL=1.2 & closest match | 13.61 | 65.7 | 34.3 | 0 |
| GID3 TL=1.5 | 4.07 | 28.4 | 12.8 | 58.8 |
| GID3 TL=1.5 & closest match | 12.45 | 66.0 | 34.0 | 0 |
| GID3 TL=2.0 | 3.51 | 28.5 | 13.0 | 58.5 |
| GID3 TL=2.0 & closest match | 12.98 | 65.6 | 34.4 | 0 |
| CID3 | 0.16 | 85.0 | 15.0 | 0 |

Legend

CT: Total classification time (seconds).
%C: Percentage of test set classified correctly.
%W: Percentage of test set classified wrongly.
%UC: Percentage of test set that could not be classified.

**Table 2: Accuracy and efficiency of induced decision trees**

creates a partition for each attribute value and then decides which irrelevant values to group together. Thus it processes each value twice. This process repeats many times when deep trees are built (e.g. when TL = 1.0). In comparison, CID3 only has to create a partition for each integer or real attribute value that occur in the training examples. Thus it processes each value only once.

## 4. Conclusion

CID3, a generalization of GID3, was described. CID3 partitions an ordered attribute's domain into two intervals ($A \leq a_i$) and ($A > a_i$). CID3 was compared with ID3, ID3-IV and GID3 in the context of generating a decision tree which distinguishes between normal and abnormal ECGs described mainly by attributes over ordered domains with large cardinality. CID3 generated the simplest decision trees with far greater predictive power than any of the decision trees generated by the other algorithms.

Future research will focus on testing CID3 on other data sets and developing more efficient algorithms for partitioning an ordered valueset into more than two intervals.

## References


[1] K P Bogart, [1983], *Introductory Combinatorics*, Pitman Publishing Inc., Marshfield, Massachusetts.

[2] J Cheng, U M Fayyad, K B Irani and Z Qian, [1988], Improved decision trees: A generalized version of ID3, *The Proceedings of the 5th International Machine Learning Workshop*, Morgan Kaufmann Publishers Inc., Los Altos, CA, 100 - 106.

[3] E B Hunt, J Marin and P J Stone, [1966], *Experiments in Induction*, Wiley, New York.

[4] J R Quinlan, [1983], Learning efficient classification procedures and their application to chess end games, *Machine Learning: An Artificial Intelligence Approach*, R S Michalski, J G Carbonell, and T M Mitchell (Eds.), Morgan Kaufmann Publishers Inc., Los Altos, California, 463 - 481.

[5] D J Rowlands, [1981], *Understanding the Electrocardiogram: A New Approach. Section 1: The Normal ECG*, Imperial Chemicals Industries PLC, Pharmaceuticals Division, Alderley Park Macclesfield, Chelshire, England.



**Acknowledgements:** The authors thank the anonymous referees for constructive comments which improved the quality of the final paper.

# Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as a Communications or Viewpoints. While English is the preferred language of the journal papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted **in triplicate** to the editor.

**Form of Manuscript**

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.
- References should be listed at the end of the text **in alphabetic order** of the (first) author's surname, and should be cited in the text in square brackets. References should thus take the following form:

  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.

  [3] S Ginsburg, [1966], *Mathematical theory of context free languages,* McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may be provided in one of the following formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- as a **WordPerfect**, **TEX** or **LATEX** or file; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies. A styles file is available from the editor for Wordperfect, **TEX** or **LATEX** documents.

**Charges**

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

These charges may be waived upon request of the author and at the discretion of the editor.

**Proofs**

Proofs of accepted papers will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

**Letters and Communications**

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

**Book reviews**

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

**Advertisement**

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 4, March 1991
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 4, Maart 1991
ISSN 1015-7999

# Contents