AN APPLICATION OF INDUCTIVE INFERENCE TO DETERMINING COMPUTER ARCHITECTURE AND COMPILER STRUCTURE

by

R. S. Michalski

Internal Report, Department of Computer Science, University of Illinois, Urbana,
February 1981

Internal Report, Department of
Computer Science, University of
Illinois, Urbana, February 1981.

ABSTRACT

An Application of Inductive Inference to Determining Computer Architecture and Compiler Structure

R. S. Michalski - February 1981

Problem description

Professor Kuck with his collaborators have developed a software system PARAPHRASE, which can analyze the performance of different computer architectures and compiler structures in executing Fortran programs. The existing data base of these Fortran programs include approximately 1000 programs which can be classified into 20-50 packs of related programs.

Each program can be run with different settings of parameters. There are typically several parameters, each of which can assume a few different values (between 2 and 10). Let P_1, P_2, ... , P_L denote the parameters of a given program. In addition, each program can be characterized by a few tens of major characteristics. These characteristics, denoted C_1, C_2, . . . , C_1, are discrete variables ranging over several values ($\leq$ 10).

The program analysis system can be applied to a given FORTRAN program using one of several computer architectures, A_1, ..., A_a (a = 8.10) and one of several compiler structures C_1, C_2, . . . , C_c (c = 5.7).

A run of the analysis system on a given program produces a set of 25 - 200 preformance measures, m_1, m_2, . . . m_u (u = 25...200) which characterize the performance of the assumed computer architecture, A, and the compiler structure C_i from the viewpoint of executing the given program.

A performance vector m_1, ... , m_u can be classified into several categories of quality, Q_1, Q_2, . . . , Q_t. In the simpliest case we can have just two categories Q_g - a good performance, and Q_6 - a bad performance.

The experiments with the program analysis have produced a very large data base of the results which describe the running the system on available programs (= 10000 programs) under different settings of parameters, and with different computer architectures and compiler structures. However, the complexity of the data is so large that it is very difficult to determine general relationships which relate the parameter settings and different program characteristics to architectures and compiler structures producing the best performance.

A mathematical analysis of this problem is possible only for some small subproblems and special cases.

A general solution of this problem has a great practical importance. An attractive possibility altering such a solution is to apply techniques of inductive inference developed in artificial intelligence. To show how it could be done, we will formulate the problem in the following way.

Let R denote an <u>event</u>, which is a vector of program parameters and program characteristics: $(P_1, P_2, C_1, C_2, \ldots, C_2)$
Suppose this program was run using a computer architecture A_i and computer structure C_j, and obtained performance vector was $m_1, m_2 \ldots, m_u$, classified as Q_g (a 'good' performance). We will denote this as

$$R \ ::> (A_i, C_j)_g$$

If the obtained performance for event R were 'bad', then we would denote this as: $e \ ::> (A_i, C)_b$
Let E^g_{ij} and E^b_{ij} are sets of events, respectively such that

$$V \ e \ e \ E^y_{ij}, \ e \ ::> (A_i, C_j)_9$$

$$V \ e \ e \ E^b_{ij}, \ e \ ::> (A_i, C_j)_b$$

The number of (A_i, C_j), using terminology of inductive learning, E^9_{ij}

and E^{b}_{ij} can be viewed as sets of 'positive' and 'negative' learning examples (i.e., examples of 'good' and 'bad' use of computer architecture A_i and compiler structure C_j for programing which we characterized by events in E^{9}_{ij} and E^{b}_{ij}, respectively).

The inductive learning methology can be thus applied here to determine the general patterns characterizing 'good' 'bad' architectures and compilers for different programming tasks.

Such a methodology has been developed over last several years by research groups of D. Michalski [___ ___ ___], and successfully applied to various inductive tasks. The most notable application is the inductive learning of Dcision rules diagnosis for soybean diseases diagnosis from examples of expert's dianostic decisions. These rules have out performed the rules representing the experts diagnostic knowledge [ ]

As was indicated before, the typical number of different architectures (a) and computer structures (c) applicable for a given set of programs is relatively small (a ≃ 8..10, C = 5.7). Therfore, one can formulate somewhat more machine learning problem:

Given: 'sets of 'positive' assertions:

$(e ::> (A_1, C_1)_g)$ where $e \in E_{11}$

$(e ::> (A_1, C_2)_g)$, where $e \in E_{12}$

$(e ::> (A_k, C_\ell)_g)$, $e \in E_{12}$

$(e ::> (A_k, C_\ell),)$, e f E_k, L

(thus

- the set of 'negative' assertions

$$\bigcup_{i_1 j} (e ::> (A_i, C_j)_b), \text{ where } \in E_{i_1 j}$$

$i = 1, \ldots, a$

$j = 1, \ldots, c$

Determine:

A set of rules

$$R_{11} ::> (A_1, C_1)_g$$
$$R_{12} ::> (A_1, C_2)_g$$
$$\vdots$$
$$R_{ac} ::> (A_a, C_c)_g$$

when R_{ij} is a logical expression specifying the relevant program characteristics and approximate parameter setting for which the computer architecture A_i and compiler C_i which give a good performance.

The above is one of possible ways of formulating the problem of selecting the computer architecture and compiler structure as an inductive task.