ELSEVIER

Information & Management 27 (1994) 89–101

INFORMATION & MANAGEMENT

Research

Expert systems shell benchmarks: The missing comparison factor

Robert T. Plant *, Juan P. Salinas [1]

Department of Computer Information Systems, University of Miami, Coral Gables, FL 33124, USA

Abstract

This paper develops a methodology for benchmarking knowledge-based systems that practitioners may use to perform a comparative analysis of expert system shells. A program utilizing a deliberate instructional mix was found to be the most suitable and accurate way to measure the value of shells. The benchmark is intended for rule-based shells, such as CLIPS, VP-Expert and Ibis. The methodology for the approach is a generic rule-based algorithm that is easily adaptable to meet the language requirements of individual shells. We present results for three shells.

Key words: Benchmarking; Expert system shells; Expert systems; System performance

1. Introduction

The concept of expert systems [12] has existed since the mid-sixties, however it was not until the eighties, when the microcomputer revolution occurred, that this technology was made easily accessible in a business environment [14]. Expert system technology has two levels application programs (e.g., MYCIN [13]) and tools (the "shells", by which applications are developed and run e.g. EMYCIN [15]). Though there are many expert system shells available, there is no measurement technique that is generally used to compare their performance. We can find out if one expert system shell offers more capabilities than another, but we cannot determine from the packaging that comes with the system how well any given shell could process a standardized problem. Inability to judge the performance of a shell can prove very frustrating, especially to those people who are first time purchasers/users of this software and have no alternative system with which to make a comparison, or for users such as NASA who run a large number of systems upon their shells, and therefore where a saving in processing speed over all systems could be advantageous. It is the intent of this paper to fill this gap by developing a methodology to benchmark expert system shells.

2. Expert systems shells' benchmark strategy

The following seven step procedure is used to create a benchmark system for expert system shells. First, the target area is defined, this is the

[1] *Acknowledgement*: The authors wish to thank the anonymous referees and Professor Sibley for their useful comments.

SSDI 0378-7206(94)E0057-B

environment in which the benchmark is to be utilized. The constraints for the benchmark are then laid down; these are the "benchmark goals," which cover aspects such as accuracy, sensitivity, etc. Having determined these parameters, the techniques by which the benchmark is to be constructed is selected, along with the statistical approach that will be used to analyze the results of performing the experiments. Having determined the framework in which the benchmark is examined, the standardized benchmark program is designed and implemented. Finally, the benchmark programs are run and the analysis performed.

2.1. Target area definition

The *benchmark* will be designed to measure the execution speed of development tools under the workload imposed by a typical business expert system. The shell will be running on an IBM PC or compatible machine with a version of the MS-DOS operating system and a hard disk drive. In order to measure the performance of our target area, we must first determine its relationship to the other components of the system. We also need to understand the effect that other factors have on the target area [9]. Obviously, the benchmark will be influenced by the speed of the compiler or interpreter; in our case this is the shell program. In turn, the compiler or interpreter's performance is influenced by the operating system. The operating system (MS-DOS) and the machine (IBM PC) remain constant throughout the experiment for all shells in order to minimize test variance. The benchmark program design is standardized, such that all shells use as small a set of operating system features as possible, minimizing bias. However, without access to the shells' source code, a complete determination of the operating system shell dependence cannot be made. In turn, we note that software performance is influenced by hardware performance, which in turn, is composed of processing speed and input/output performance. The benchmark may be run on different types of PC compatible computers. So, we must provide a way to assess the differences in processing speed and hard disk performance produced by the differences in hardware characteristics. We need this assessment to relate the results of running the benchmark programs on different computers.

Here, we discuss the effect of running the benchmark programs on three different machines: a 10 MHz XT compatible, an 80386SX 16 MHz AT compatible, and an 80386DX 25 MHz AT compatible machine. Future shell benchmark programs may be written for new development tools, using the specifications produced here; these new programs may be run on different machines. The hardware benchmark programs will provide a way of relating future results to those discussed here, since hardware characteristics may be compared using the results of the commercial hardware benchmark programs.

2.2. Goals of the expert system shell benchmarks

After defining the target area, the goals of the benchmark must be set in terms of accuracy, sensitivity, portability, and effort [3][4]. We begin with *accuracy*. A goal of the benchmark is to provide sufficient accuracy to guarantee that if it takes a shell less time to execute the benchmark than another, it is because that shell is faster for many business applications. However, there may be expert systems that, due to the nature of their algorithms, run faster on the shell though the benchmark program ran slower. This is because the benchmark is based on an estimated average of the typical business expert system's workload. In order to obtain the desired level of accuracy, we must be careful in applying five points. First, select the contents of the benchmark programs carefully, so that it represents the work load imposed by a typical business expert system. Second, be clear and concise in preparing the benchmark specifications to avoid any programmers' interpretations and abilities that produce inaccuracies. Third, analyze and standardize the contents of the system files of the machines (e.g., possibly empty all other directories) used to run the benchmarks; system files are invoked in all operations. Fourth, to guarantee the desired level of accuracy, consider problems introduced by any lack of compatibility among different development tools. For example, differences in language

semantics can be a source of inaccuracies. Most development tools provide a syntax to create production systems; the production rules should follow a standard structure, as depicted in Fig. 1. To avoid possible implementation inaccuracies, the benchmark only includes production rules.

The fifth consideration, involves the method of measuring the execution time of the benchmark programs.

We also have to define goals in terms of *sensitivity*. The benchmark must be able to show major differences in execution speed of the shells. It must be sensitive to those instructions most commonly used by business expert systems. To guarantee this, we perform a thorough statistical analysis of the selected variety of expert systems.

Because of the differences in syntax of the development tools, *portability* is an important issue. By using only production rule systems, the syntax becomes almost standard with a few variations due to wording and termination characters. This makes the benchmark portable; translating the benchmark involves modifying the syntax only slightly.

It is also important to provide for variety in control commands. An expert system may be modelled in two portions: control code and rules. The *control code* tends to be different from one shell to another. For this reason, we try to keep the control architecture of the benchmark as small as possible, while attempting not to bias the experiments in any way.

The *effort* can be subdivided into two parts:

(i) The design and statistical analysis that produces the benchmark specifications, and
(ii) Translation of the benchmark into actual tests.

We try to minimize the second by producing a clear specification and minimizing the control code. In addition, we reduced the translation effort by developing a rule generator program that can alter the code of the program to comply with the system under test.

2.3. Benchmark technique selection

The third step in developing a benchmark is to select a technique and methodology applicable to the considered target. Several benchmarking techniques are available, each having its own advantages and disadvantages. Instruction mix, arbitrary program, artificial workloads and application program [1][5][17] are amongst the best known.

After analyzing the targets characteristics, we found that the instructional mix technique best met the parameters of our benchmark requirements in terms of accuracy, sensitivity, portability, and effort.

2.4. Statistical analysis

Statistical analysis is a crucial component of benchmark design. Results establish the link between the benchmark and the target application. We attempted to ascertain the internal composition of several business expert systems and to determine what was a "typical business expert system". The design of our statistical analysis involves selecting the type and focus of these statistics.

2.5. Type of statistics

Two statistical analysis options need to be considered: *dynamic* and *static*. Static analysis produces a statistical perspective of the components

```
IF condition
THEN action_a
[ELSE action_b].
```

The condition is also known as the left hand side of the rule (LHS);
action_a and action_b are known as the right hand side (RHS).
The ELSE clause is optional.

Fig. 1. Syntax of IF/THEN/ELSE rules.

of a system, such as the number of assignment statements, operations, loop control statements, and other instructions contained in the code of the program. Dynamic analysis reflects the actual behavior of the program, achieved through counting the number of times each type of instruction is executed rather than just counting the number of occurrences, in the program.

To obtain accurate profiles of the business systems that we were using as the basis of our benchmark programs, we performed both dynamic and static analysis, through a technique called *Real Problem Execution Simulation (REPROES)*, in which a manual procedure was performed to execute a series of case studies on actual business expert systems.

2.6. Shell internal functioning

After determining the type of statistics desired and the method to compile them, we had to define those characteristics of expert systems we wished to study. Every characteristic examined must have an impact on the execution speed of expert systems and must represent the vast majority of the running time of the programs. The two key components in the inferencing procedure are the *chaining mechanism* and *pattern matching*.

2.7. Focus of the statistical analysis

Pattern matching is a key element for both forward and backward chaining methods. Thus, composition of patterns is a major element in the systems performance. However, the literature covering the relationships of performance to pattern composition or size is weak.

The importance of this can be seen by running several tests on VP-Expert, Ibis, and CLIPS [it should be noted that these shells were selected arbitrarily and the benchmarks produced are not intended to be a reflection of the commercial suitability of the products. The paper present a benchmark methodology and uses these shells only to demonstrate the technique]. The tests were to determine if the length of the patterns made any difference in the execution time. Pattern length refers to the number of ASCII characters composing each pattern. The knowledge bases of these tests did not include any rules. Their only function was to assert 200 facts to the cache memory; that is, to relate one pattern to another. The syntax of Ibis and VP-Expert's assertions are very similar: pattern_a = pattern_b.

The syntax of CLIPS is different: DEFFACTS (assertion_name(pattern_apattern_b)).

Although both instructions may seem differ-

Pattern Assertions

Average Execution Time Expressed in Seconds (a)				
Pattern Lengths	25 X 25	25 X 2	2 X 25	2 X 2
CLIPS	1.1515	0.9075	0.9095	0.6645
VP-Expert	9.5665	9.5675	1.9625	1.9485
Ibis	3.9610	3.9570	1.4470(b)	1.4340(b)

(a) These times are the average of 20 executions running on a 80386DX 25 MHz.
(b) Ibis requires variable names to be at least 3 characters long

Percentage Changes From Results of 25 X 25				
Pattern Lengths	25 X 25	25 X 2	2 X 25	2 X 2
CLIPS	0.00%	-21.19%	-21.02%	-42.29%
VP-Expert	0.00%	0.01%	-79.49%	-79.63%
Ibis	0.00%	-0.10%	-63.47%	-63.80%

Fig. 2. Pattern assertion execution times.

ent, they have the same purpose: to make a logical connection between the two patterns. Fig. 2 contains the results of these tests.

Although the number of assertions was kept constant, decreasing the size of the patterns produces reductions in run-time of up to 80%. Though the programs only tested assertion efficiency, they proved that number of instructions executed is not the only factor to be observed. The length of the patterns used by business expert systems must also be examined, allowing us to determine the average length of the patterns used by most business expert systems. The shell's efficiency in pattern matching and the size of these patterns will have a tremendous effect on the execution speed of expert systems.

It is *important* to note that the tests mentioned above are not benchmark programs. They are intended to demonstrate the differences in execution times produced by variations in patterns' lengths, rather than demonstrating the speed of the shells. These tests were performed by creating sample knowledge bases through program generators.

Rules stored in the knowledge base is the next issue. First, the total number of rules must be counted. The more rules, the greater the search space for the inference engine. We also must count how many of these rules were actually executed. In backward chaining, we count rules that fail as fired rules. That is, if the inference engine fails in an attempt to fire a rule, it will have to perform a second search for another rule.

The final factor to observe is the composition of the rules fired. The rule components are divided into those included in the LHS and those in the RHS. In the rules that fail, we will record those elements of the RHS that were executed as the only components of that rule. This must be done because our benchmark will run using both forward and backward chaining. The workload must be identical for both methodologies. Instead of having rules that fail, we increase the number of rules to be executed. This produces a similar effect on the benchmark's workload. To emphasize, it is critical to note the characteristics of the patterns used, particularly because this will have a great impact on the speed of the conflict resolu-

DYNAMIC STATISTICS(a) Dynamic Behavior of a Typical Business Expert System	
Number of Rules	
Rules in the knowledge base:	53 rules total
Number of rules fired:	44 rules total (b)
Number of rules fired from within another rule:	25 rules (c)
Average depth of an intermediate chain:	3 rules (d)
Composition of the LHS	
string_pattern_a = string_pattern_b:	50 comparisons total
length of string_pattern_a:	6 characters average
length of string_pattern_b:	6 characters average
integer ¦('<' '>' '<=' '>=' '<>')'¦ integer:	11 comparisons total
real ¦'<' '>' '<=' '>=' '<>')'¦ real:	14 comparisons total
AND's:	13 instances
OR's:	9 instances
Composition of the RHS(e)	
string_pattern_c = string_pattern_d:	57 string assertions total
pattern = real_number:	9 real number assertions
pattern = integer_number:	7 integer assertions total
length of string_pattern_d:	6 characters average
pattern_e = pattern_f ¦('+' '-')'¦ pattern_g:	8 computations total(f)
pattern_h = pattern_i ¦('x' '/')'¦ pattern_j:	6 computations total(g)

Fig. 3. Resulting data from the statistical analysis.

tion process. All types of comparisons and assertions must be recorded, including the type and size of the elements being compared or stored. Although development tools offer many other functions such as I/O, graphics and spreadsheet interfaces, these tend to be nonstandard from shell to shell. We therefore concentrate on the real burden of expert system processing: conflict resolution and inference.

We next examined six business expert systems in operation through the REPROES technique. The results of this systematic study are compiled in Fig. 3, where the letters between brackets refer to the following:

(a) The results of this data may vary from execution to execution, since they are based on actual behavior of the programs tested. The path followed to reach a solution changes with the problem presented to the expert system. This produces a different program behavior for each different problem solved. The results are the averages obtained by running each expert system several times, following the REPROES technique.

(b) This result includes partially fired rules of the backward chaining expert systems tested.

(c) This result counts for the rules that are not fired from the control architecture but from within another rule, either by a FIND statement or during the inference process.

(d) An intermediate chain occurs when a rule calls another rule. The second rule calls a third rule, and so on. This result measures the average number of rules called.

(e) The RHS of the rules fired may contain additional statements not considered here. These numbers only represent the data considered relevant to the benchmark design.

(f, g) Multiple operator statements count as multiple statements. For example, the statement pattern = A × B + C counts as two statements, one multiplication and one addition.

2.8. Benchmark design

After determining the actual behavior of a typical business expert system, we proceed to the design stage of the benchmark. This step produces the program specifications. These are based on the dynamic statistics. Fig. 4 includes the benchmark program specifications. The imple-

Benchmark Program Specifications
The benchmark programs must have 53 rules, and only 44 out of those 53 rules will get fired during execution. 25 of the 44 rules will be executed from within other rules, and the rest must be called from the control architecture. The average depth of the intermediate chains must be 3 rules. There will be 50 string comparisons. The benchmark program must have had a total of 13 AND and 9 OR statements in the LHS of fired rules. Also, there must occur 57 string comparisons during the execution of the program. The average length of all string patterns must be six characters. There will be 8 additions and subtractions altogether, and 6 multiplications and divisions altogether, as well. The number of additions must be the same as the number of subtractions. This also applies to the number of divisions and multiplications. All string patterns used throughout the program must be different. This applies to rule names, patterns, and contents of the variables. Each rule should have a comment, stating the rule number. The program should not use any type of display or print statement.

Fig. 4. Benchmark program specifications.

mentation of the benchmark programs must follow these specifications closely.

2.9. Implementation of the benchmark programs

We attempt to benchmark three development tools: VP-Expert Version 2.1 by Paperback Software International, CLIPS version 4.2 developed at NASA/Johnson Space Center, and Ibis educational version 4.33 by Intelligence Manufacturing Company [2][8][16]. VP-Expert and Ibis use backward chaining, whereas CLIPS utilizes forward chaining.

Benchmark programs were produced through our code generator, which itself was written in Clipper Version Summer 87. However, due to the divergence in logic between forward and backward chaining, the benchmark for CLIPS varies slightly from the other two benchmark programs. The first difference is that the contents of the LHS and the RHS had to be altered. The result of running both sets of rules will be the same, and the processing requirements for both is very similar. In addition, CLIPS requires an additional rule that invokes all other rules. This rule starts the program by making the necessary pattern assertions to the cache memory required by the other rules. Even with these differences, the burden imposed by the CLIPS' benchmark program is identical to the Ibis and VP-Expert programs within our accuracy constraints.

2.10. Timing of the benchmark programs

Upon developing the benchmark programs, we must devise a technique that guarantees precision and fairness in measuring execution times. The first alternative is to use the internal clock of the computer. The major limitation of this approach is that the internal clock only counts in seconds, so lack of precision rules out this option. Therefore, we must use an external chronometer. The new problem is therefore: how can we start and stop the external timer? Because human response time is slow and inconsistent, manual control is not effective. We need to create an interface between the computer and the timer that is accessible from all expert system programs. In addition, portability becomes an issue.

As a result, we created a purpose built timing-device: The "Chronosound". As its name suggests, this device is a chronometer that works with sounds. The instrument is connected to the terminals of the internal speaker of the computer. It contains an internal audio amplifier that transforms the speaker output into the necessary voltage needed to activate a micro relay. This relay, in turn, controls an external chronometer. Fig. 5 shows the circuit diagram of the Chronosound.

In testing the measuring device, we discovered that different computers send different output voltage levels to their speakers. To amend these discrepancies, we adjust the audio level through the audio gain control. The frequency and duration of the sound that starts and stops the chronometer may need adjustment and this is determined through a program that tests different frequencies and durations, so that the optimum signal for a given computer can be identified. After obtaining the frequency and duration of sound, we use a small program "START-STOP.EXE" that contains the instruction "sound-frequency, duration", (frequency and duration contain the values obtained before). Running this program will cause the Chronosound to start or stop the chronometer, depending on its previous status.

To use the Chronosound with our benchmark

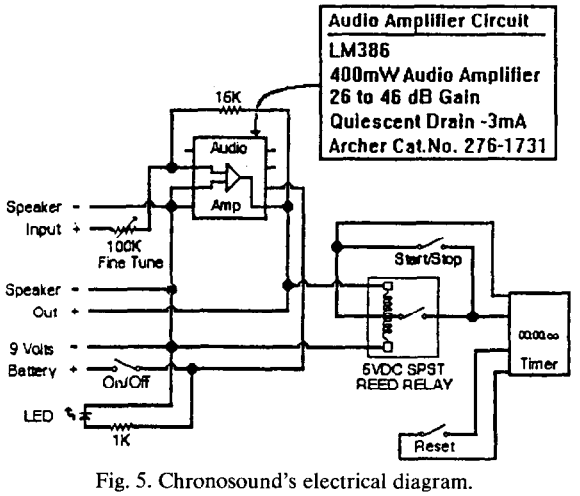


Fig. 5. Chronosound's electrical diagram.

	PC-XT		80386 SX		80386 DX		
	4.77 Mhz	10 Mhz	10 Mhz	16 Mhz	8 Mhz	24 Mhz	
	Expert Systems Benchmark Programs						
Clips	6.396	3.006	1.454	1.130	1.639	1.187	Seconds (a)
Ibis	59.250	41.450	15.212	6.013	11.906	3.686	Seconds (b)
VP-Expert	11.400	5.615	2.234	1.698	2.315	0.985	Seconds (c)
	Benchmark Program V. 1.20 by Chips and Technologies Inc.						
Overall Performance	0.21	0.42	1.21	1.84	1.03	3.36	MIPS (d)
	CheckIt Benchmark Programs						
CPU Speed	344	689	1,674	3,157	1,842	6,374	Dhrystones (e)
Video Speed	465	762	3,543	5,212	3,277	6,995	Characteres/Second (e)
Math Speed	6.5	13.6	31.2	60.4	35.4	121.5	Kilo-Whetstones (e)
Average Seek Time	64.1	64.6	19.8	19.8	21.2	20.7	Milliseconds (f)
Track to Track Seek Time	8.8	8.8	6.0	6.0	1.0	1.0	Milliseconds (f)
Transfer Speed	28.2	100.1	594.3	494.3	514.0	514.0	Kilobytes/Second (f)
	CORE Disk Performance Test Program V. 2.8						
Disk Capacity	32.7		80.3		89.0		Megabytes (f) (g)
Number of Cylinders	939		922		1,023		Cylinders (f) (g)
Number of Heads	4		5		10		Heads (f) (g)
Number of Sectors per Track	17		34		17		Sectors (f) (g)
Transfer Speed	28.1	189.1	670.5	686.2	807.0	980.0	Kilobytes/Second (g)
Average Seek Time	59.6	62.7	6.1	20.1	21.2	21.0	Milliseconds (g)
Track to Track Seek Time	16.6	17.0	6.1	6.2	1.2	1.0	Milliseconds (g)
Performance Index	1.090	2.003	6.741	6.823	7.396	8.458	Benchmark Units (g)
	Norton Utilities V.5.0 System Information Benchmarks						
CPU Speed	1.0	2.0	4.9	8.9	5.9	22.5	Benchmark Units (h)
Disk Speed	0.8	1.5	6.3	6.3	6.6	7.1	Benchmark Units (i)
Average Seek Time	62.95	62.83	19.72	19.71	20.87	20.89	Milliseconds (i)
Track to Tracl Seek Time	9.81	9.93	5.17	5.16	3.62	3.51	Milliseconds (i)
Transfer Speed	29.1	142.7	711.9	711.1	780.9	878.1	Kilobytes/Second (i)
Overall Performance Index	0.8	1.8	5.3	8.0	6.1	17.3	Benchmark Units (j)

Fig. 6. Benchmark results.

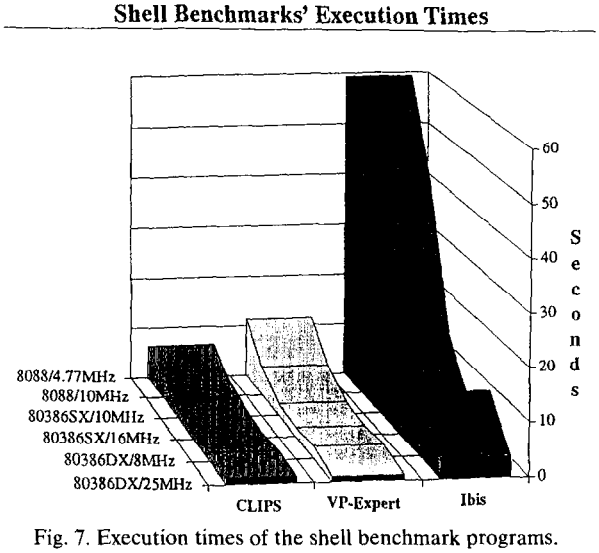


Fig. 7. Execution times of the shell benchmark programs.

programs, we include two system calls within the benchmarks to the program START-STOP.EXE. The first call must be placed at the beginning of the expert system and the other at the end.

The Chronosound was used in a way to guarantee that all times are taken under the same conditions. The results produced by the Chronosound have a 0.03 seconds variance. The average

```
8088 -    4.77 MHz:  1.0
8088 -      10 MHz:  2.0
80386 SX - 10 MHz:  4.9
80386 SX - 16 MHz:  8.9
80386 DX -  8 MHz:  5.9
80386 DX - 25 MHz: 22.5
```

Fig. 8. Norton's CPU benchmark results.

```
Average = 7.333
Standard Deviation =  7.174

(1.0-7.333)/7.174) = -0.911
(2.0-7.333)/7.174) = -0.771
(4.9-7.333)/7.174) = -0.367
(8.9-7.333)/7.174) =  0.191
(5.9-7.333)/7.174) = -0.228
(22.5-7.333)/7.174) = 2.086
```

Fig. 9. Computations to produce Fig. 10.

time for 40 executions of the shell benchmark programs on each computer were calculated. The number of executions was chosen to obtain the execution times within 0.015 seconds with 98% confidence or better. Using the standard sample size formula the actual result calculated was 22 samples or executions; but by increasing the number of samples to 40, we also increase the confidence factor. By measuring 40 executions, we are at least 98% confident that the margin of error is not greater than 0.015 seconds.

Since the three computers used to run the benchmarks allowed is to select the processor's speed, all benchmarks were run at both speeds available on each machine. Fig. 6 contains the results obtained from all the benchmarks, including the shell and the hardware benchmarks.

With all the benchmark programs run and compiled, the results of the benchmarks can be compared to determine the fastest shell. These results are diagramed in Fig. 7.

Combined Benchmarks Results

In Number of Standard Deviations from the Mean

Grouped by Benchmarks' Target Areas

	XT/ 4.77Mhz	8088 10Mhz	AT/ 10Mhz	80386 SX 16Mhz	AT/ 8Mhz	80386DX 25Mhz
Clips (a)	2.106	0.288	-0.544	-0.718	-0.445	-0.687
Ibis (b)	1.780	0.908	-0.378	-0.828	-0.540	-0.942
VP-Expert (c)	2.045	0.437	-0.502	-0.651	-0.480	-0.849
Average (**)	1.977	0.544	-0.475	-0.732	-0.488	-0.826
CPU Speed (e)	-0.994	-0.823	-0.334	0.402	-0.251	1.999
CPU Speed (h)	-0.911	-0.771	-0.367	0.191	-0.228	2.086
CPU Speed (d)	-1.085	-0.884	-0.129	0.473	-0.301	1.926
Math Speed (e)	-0.997	-0.812	-0.353	0.407	-0.244	1.999
Average (**)	-0.997	-0.823	-0.296	0.368	-0.256	2.003
Transfer Speed (i)	-1.559	-1.214	0.515	0.513	0.725	1.020
Transfer Speed (f)	-1.555	-1.232	0.990	0.540	0.629	0.629
Transfer Speed (g)	-1.573	-1.097	0.326	0.373	0.730	1.242
Average Seek Time (*)(i)	-1.411	-1.409	0.794	0.796	0.617	0.614
Average Seek Time (*)(f)	-1.407	-1.414	0.792	0.792	0.582	0.654
Average Seek Time (*)(g)	-0.802	-0.819	2.143	-0.142	-0.194	-0.185
Track to Track Seek Time (*)(i)	-1.228	-1.244	0.022	0.027	1.153	1.271
Track to Track Seek Time (*)(f)	-0.771	-0.771	-0.641	-0.641	1.412	1.412
Track to Track Seek Time (*)(g)	-0.829	-0.833	-0.560	-0.567	1.178	1.611
Disk Speed (i)	-1.538	-1.267	0.595	0.595	0.711	0.905
Disk Performance Index (g)	-1.542	-1.217	0.471	0.500	0.705	1.083
Average (**)	-1.292	-1.138	0.495	0.253	0.750	0.932
Overall Performance Index (j)	-1.065	-0.879	-0.231	0.268	-0.083	1.990

Fig. 10. Number of standard deviations away from the mean.

An analysis of the results in Fig. 7 shows us that Ibis performance is far from satisfactory. It took the shell almost a minute to run the benchmark on a 4.77 MHz PC, more than nine times CLIPS's run time, and five times VP-Expert's run time. The Ibis requires a powerful computer to provide an acceptable response time. CLIPS, on the other hand, proved to be the fastest shell. Its response time was good for almost all the machines. The only case where this execution time was not satisfactory was in the 4.77 MHz test, which took over six seconds to execute. Since this type of computer is almost extinct, this becomes less of an issue. CLIPS also produced interesting results for the 16 MHz 80386SX and the 25 MHz 80386DX. Although the first machine is faster than the second one, CLIPS runs faster on the SX. The only explanation to this phenomena is the fact that the hard disk on the SX is slightly faster than the one on the DX. Still, the difference of only five hundredths of a second is quite small, if compared to the difference obtained from the Ibis benchmarks (2.327 seconds slower on the SX than on the DX). From this one may conclude, CLIPS uses the hard disk more than VP-Expert and Ibis, since VP-Expert and Ibis behaved as expected, running faster on the faster machine. This may make shells such as VP-Expert and Ibis more competitive with faster 80486 and subsequent generation processors.

Because the values represent different types of information and are expressed with different notations, it is difficult to infer the relationships among the different variables. We created Fig. 10 to show the relationships. The values in Fig. 11 are stated as "the number of standard deviations that each value of Fig. 6 moves away from the average computed for that benchmark". This was achieved by determining the CPU speeds as reported by Norton Utilities CPU Speed benchmark, Fig. 8.

We calculated the average (7.333), together with the standard deviation (7.174). With these two figures, we can perform the computations shown in Fig. 9.

The numbers we calculated are the same as those of the line "CPU Speed (h)" of Fig. 10 although Fig. 10, does not provide the actual results of each test, it enables us to make comparisons among the different characteristics of the

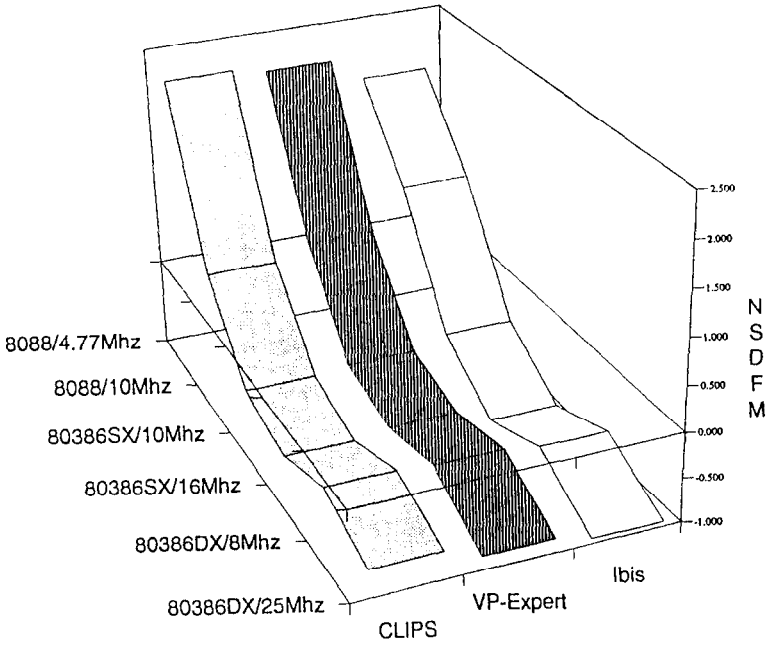


Fig. 11. Differences in shell benchmarks' execution times.

hardware and the running speed of the shell benchmark programs.

The rows in Fig. 10 have been sorted by the target area of the benchmarks. The first section includes the expert systems' benchmarks. The second section includes the benchmarks targeted at the processing speed. The third section includes the hard disk benchmarks. The last section includes the overall hardware benchmark.

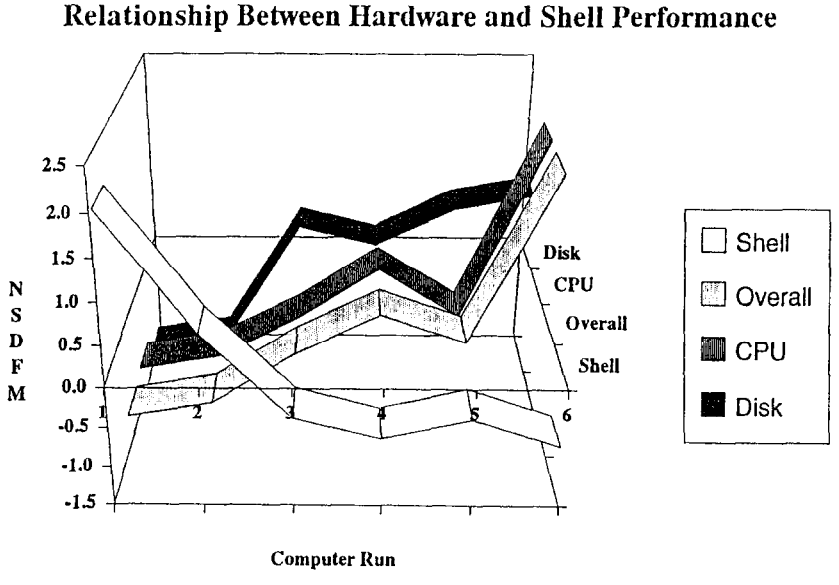
Fig. 11 provides a graphical representation of the differences in execution speed among the computer runs. The term "computer run" refers to the execution of a benchmark program on a given machine at a given speed. Because each machine allows two speeds, two runs can be obtained from each. The vertical axis indicates the "number of standard deviations away from the mean." The three plots are the three shell benchmarks.

Fig. 12 shows that the execution times for the three benchmarks denoted almost the same pattern for all six computer runs. The three lines, representing the three benchmark programs, are almost identical because the instruction mix of each program is equivalent. If this were not the case, we would not see such even behavior among the shells' execution times when switching from one computer run to the next.

Finally, computer performance is the product of several factors and hardware processing speed will have an influence on shell performance. Fig. 12, illustrates this point and provides a graphical perspective of the influence that different hardware characteristics have upon shell performance.

Fig. 12 is based on the calculated averages obtained for the four groups of benchmarks of Fig. 11: shells, processing speed, disk speediness, and overall hardware performance. The vertical axis shows the number of standard deviations that each run is away from the computed average for that type of benchmark over all the computer runs, as calculated in Fig. 10 on the rows labeled "Average (**)" and "Overall Performance Index (J)".

The four plots represent the changes in bench-



Note1: NSDFM=Number of Standard Deviations from the Mean
Note2: 1=8088/4.77MHz, 2=8088/10MHz, 3=80386SX/10MHz, 4=80386SX/16MHz, 5=80386DX/8MHz, 6=80386DX/25MHz

Fig. 12. Relationship between hardware and shell performance.

mark results from one computer run to the next for each of the four benchmark groups. We can see how the shell's benchmarks ran faster, thus taking less time to execute on faster machines that obtained higher ratings on the hardware benchmarks. Note that, the shell benchmarks' results are given in seconds, whereas the commercial hardware benchmarks produce ratings. The faster the hardware, the higher the ratings of the hardware benchmarks; the faster the shell benchmarks executed, the less time to run them.

A final observation is that differences in Disk, CPU and Overall Hardware Speed produce changes in the execution speed of the benchmarks to different degrees. As noted earlier when selecting goals, the shell benchmarks are not hardware insensitive. The results of the hardware benchmarks allow the reader to relate possible future benchmarks of other shells run on different hardware.

3. Conclusion

We attempted to develop a benchmark for shells or development tools. These benchmark programs allow for comparison of different shells based on their execution speed. In the past, such comparisons could only be made based on the features and capabilities of the different programs available. After implementing the benchmark programs, we see that CLIPS is faster than Ibis, for business applications. If we only look at the features and capabilities of the programs, Ibis would probably be the winner, because it provides excellent development and user interfaces. In addition, Ibis offers both forward and backward chaining, while CLIPS only permits forward chaining.

In the past, researchers have attempted to develop benchmark programs for expert systems. Their major flaw was that they did not establish the link between their benchmark programs and the real world. People developed expert systems to compare different tools, but only to a certain degree. Although we can use an expert system that solves the "monkey and the bananas" problem to compare shells [6], we cannot conclude that a shell that runs this program in less time will also run business applications faster. The second error was that they did not utilize the fact that expert system performance is strongly related to its pattern matching capability. Developing a set of rules that call each other and count the number of rules fired does not tell us anything about the shell unless we know the contents of the rules and base those contents on some logical criteria.

This paper summarizes the efforts of many people, who for over 20 years, have studied benchmarks. Only a few researchers have attempted to develop a benchmark program for expert systems, and their efforts tended to be "one-off" attempts; e.g., [7][10][11].

References

[1] Bell, A.G., Hallowell, and Long, D.H., Software-Practice and Experience, "A Universal Benchmark?" October 1973, P. 355
[2] CLIPS Version 4.2 (NASA-Johnson Space Center)
[3] Conte, Thomas M. and Hwu, Wen-mei, Computer, "Benchmark Characterization," January 1991, P. 49.
[4] Curnow, H.J. and Wichmann, B.A., The Computer Journal, "A Synthetic Benchmark," February 1976, P. 43
[5] Ferrari, Domenico, Computer, "Workload Characterization and Selection in Computer Performance Measurement," August 1972, P. 20
[6] Giarratano, Joseph and Riley, Gary, Expert Systems: Principles and Programming (Boston-Massachusetts: PWS-KENT Publishing Co., 1989), P. 160.
[7] Gilbreath, Jim, BYTE, "A High-Level Language Benchmark," September 1981, P. 180
[8] Ibis Educational Version 4.33 (West Sacramento-California: Intelligence Manufacturing Company)
[9] Levy, Henry M. and Clark, Douglas W., Computer Architecture News, "On the Use of Benchmarks for Measuring System Performance," December 1982, P. 5
[10] Press, Larry, AI Expert, "Eight-Product Wrap-Up," September 1988, P. 64.
[11] Press, Larry, IEEE Expert, "Expert Systems Benchmarks," Spring 1989, P. 37–44.
[12] Rich, E., Artificial Intelligence, 1983. McGraw Hill, New York.
[13] Shortliffe, E.H. Computer-based Medical Consultations: MYCIN, Elsivier, New York.
[14] Silverman, B.G., 1987. Addison-Wesley, Reading, MA.
[15] van Melle, W., Scott, A.C., Bennett, J.S.,&Peairs, M.A., The EMYCIN Manual. Technical Report, Heuristic Programming Project, Stanford University, 1981
[16] VP-Expert Version 2.1 (Oakland-California: Paperback Software International)

[17] Walters, R.E., The Computer Journal, "Benchmark techniques: a constructive approach." February 1976, P. 50

Robert T. Plant, is an Assistant Professor in Computer information Systems Department, School of Business Administration, at The University of Miami, Coral Gables, Florida. Dr Plant obtained his Ph.D in Computer Science at The University of Liverpool, England. Previously having studied at The Programming Research Group, Oxford University, England. His research interests are in Software Methodology, Formal Methods, Software Economics and Metrics. Dr Plant is also a Chartered Engineer and a Senior Member of the American Institute of Aeronautics.

Juan Salinas is a Systems Consultant in Costa Rica, Central America. He has a Master of Science Degree from the School of Business Administration at The University of Miami, Florida in Computer Information Systems and a Bachelor of Science in Computer Information Systems from Bentley College, Waltham, Massachusetts.