Run–Time Prediction for Production Systems

Franz Barachini, Hans Mistelberger
Alcatel–ELIN Research Centre
Ruthnergasse 1–7
1210 Vienna, AUSTRIA
Tel.: (+43) 222 / 39 16 21 / 150
Email: es_barac@rcvie.at

Anoop Gupta
Department of Computer Science
Stanford University
Stanford, CA 94305, U.S.A.
Tel.: (+1) 415 / 725 37 16
Email: ag@pepper.stanford.edu

Abstract

A major obstacle to the widespread use of expert systems in real–time domains is the non–predictability of response times. While some researchers have addressed this issue by optimizing response time through better algorithms or parallel hardware, there has been little research towards run–time prediction in order to meet user defined deadlines. To cope with the latter, real–time expert systems must provide mechanisms for estimating run–time required to react to external events.

As a starting point for our investigations we chose the RETE algorithm, which is widely used for real–time production systems. In spite of RETE's combinatorial worst case match behavior we introduce a method for estimating match–time in the RETE network. This paper shows that simple profiling methods do not work well, but by going to a finer granularity, we can get much better execution time predictions for basic actions as well as for complete right hand sides of rules. Our method is dynamically applied during the run–time of the production system by using continuously updated statistical data of individual nodes in the RETE network. [1]

Introduction

An expert system operating in a real–time environment would typically need to react to interrupts quickly and to generate a response within a given time–frame (Laffey et al. 1988). In this respect, execution time variance is the primary problem in providing performance guarantees for real–time production systems.

Speeding up software and hardware is one approach to meet certain deadlines. RETE (Forgy 1982) and TREAT (Miranker 1987) proved to be fast pattern matching algorithms for production systems. Special–purpose hardware has been developed (Bahr et al. 1991; Gupta et al. 1986a; Gupta et al. 1986b; Gupta & Tambe 1988) to increase the performance of RETE. However, speed–up is limited and does not solve the run–time prediction problem. In order to have an indication whether a deadline will be met, match–time predictability is a necessity for real–time applications. This is justified by the fact that most of the overall run–time of expert systems is used in the match phase (Gupta 1986). Ideas limiting match complexity are given by Haley (Haley 1987), Wang (Wang et al. 1990), Tambe (Tambe & Newell 1988; Tambe, Kalp, Rosenbloom 1991), and Acharya (Acharya 1991). Haley and Wang try to put limits on the amount of data approaching the expert system. Tambe and Acharya shift match combinatorics from knowledge search to problem space search.

Our first step towards the long–term goal of building real–time expert systems is the prediction of the time taken by basic actions in the right hand sides of rules. For the sake of RETE's consistency (Barachini & Theuretzbacher 1988) these actions are usually implemented as non–preemptable code pieces in most production system languages. Hence, the time for the execution of such an action determines the granularity at which interrupts can actually be handled[2]. Moreover, if the estimated time for a basic action exceeds a user–defined deadline, the production system architecture allows us to raise certain exception handlers.

Our next step is the prediction of run–time for complete right hand sides. We show that this goal can be achieved with the same accuracy as for basic actions, and therefore the same deadline strategies can be applied.

By using our method, a high–level reasoning system could detect situations when rule execution time might exceed an allocated time interval. If this is the case an exception handler would be raised delivering incomplete or suboptimal results to the high–level reasoner.

The estimation method for the basic actions and the complete right hand sides is dynamically applied during the run–time of the production system by using continuously updated statistical data of individual nodes in the RETE network. The idea is not restricted to RETE but can be applied to TREAT as well.

After a brief overview on production systems, we introduce our notion of run–time predictability. We show that sim-

1. This research is part of the ESPRIT project CIM–AI and supported by an FFF and ITF grant from the Austrian government.

2. In the literature this is also known as the *Responsiveness* of a system.

ple profiling methods do not work well for run–time prediction. We describe the technique our run–time estimation tool is going to use for run–time forecast. We compare forecast and actually measured run–times on several typical production systems. The run–time estimation method will be explained in terms of PAMELA (PAttern Matching Expert system LAnguage) (Barachini 1991).

Basic Statements and Definitions

For the following discussion it is assumed that the reader is familiar with the RETE or TREAT algorithm as well as with the vocabulary normally used in reference to OPS (Brownston, Farrell, & Kant 1986; Forgy 1981).

Production Systems

A production system consists of a rule set, called the Production Memory (PM) and a database of assertions, called the Working Memory (WM). Each rule consists of condition statements, called the Left Hand Side (LHS) and a set of actions, called the Right Hand Side (RHS).

The RHS specifies information which is to be added to or removed from the WM. In PAMELA there are basically three possible actions in the RHS. MAKE adds a new Working Memory Element (WME) to WM. REMOVE deletes an existing WME from the WM. CHANGE modifies an already existing WME. Each WME corresponds to a certain type of data, called its Working Memory Type (WMT). Each RHS consists of one or several WME actions (MAKE, CHANGE or REMOVE).

The production system repeatedly performs the so–called Recognize–Act Cycle (RAC). During this cycle a RHS is executed, leading to a match phase which determines a Conflict Set (CS) of satisfied rule instantiations. The CS resolution procedure selects one rule to be fired and the act procedure executes the RHS.

The PAMELA inference engine uses an optimized RETE match algorithm (Barachini 1988). The RETE algorithm maps LHSs of rules to a discriminating data flow network. The data elements flowing through this network are called token. When a WME is added to or removed from the WM, a positive tagged token (plus–token) resp. a negative tagged token (minus–token) representing this action is passed to this data flow network. There are different node types available in this network. The constant condition tests are performed at so–called one–input nodes (1IN) and matched tokens are stored in alpha memories. Copies of matched tokens are sent to successor nodes. The most run–time intensive node normally is a two–input node (2IN). Tokens arriving at two–input nodes are compared against the tokens stored in the opposite token memory. Successfully joined tokens are then sent again to a successor node. In this manner tokens flow through the network until they arrive at the end–node. If this is the case an instantiation enters the CS.

Run–Time Predictability

Within the context of the present paper we will investigate predictability for the following quantities:

- the time needed to perform one WME action and
- the time needed to perform all WME actions of a rule.

Measurements have shown that most of the overall run–time of RHS is used by WME actions (Gupta 1986a). Therefore, it is worthwhile to concentrate our studies on these specific actions. We are aware that the RHS of a rule may include other statements than WME actions. These statements are assumed to be procedural whose run–time predictability is not the subject of this paper.

In most production system languages, WME actions are indivisible subtasks. Whenever an interrupt modifying WM contents rises during the match phase, it cannot be handled immediately. For the sake of consistency, it must be postponed and can only be executed after the completion of the current action. In PAMELA, at the end of each WME action, the system reaches a so–called preemption point. At this preemption point, any interrupt is handled which may have risen since the previous preemption point. Thus, the time required for a WME action is particularly crucial as it represents the time the system needs to react to an external interrupt.

The run–time for a rule's RHS represents the time for a complete rule firing. Predictability at this level enables the system to determine whether the rule can be executed within an allocated time frame.

Profiling Methods

An obvious approach to run–time predictability is profiling of the quantities we want to predict. In this section we will assess the quality of profiling methods with respect to the prediction for WME actions and for RHSs of rules.

As a starting point of our investigation, we measured the run–time behavior of WME actions for specific WMT's over all fired rules. We performed a whole bunch of measurements with ParaOPS5 (Kalp et al. 1988) on the Encore Multimax at Stanford. At the level of a WME action, the results depend on the WMT considered. For some WMT's the run–time for a WME action is nearly constant, for other WMT's run–time exhibits irregular behavior. By irregular run–time behavior we mean that the time for the execution of the WME action differs substantially from one RAC to the next.

In order to find meaningful data even in this latter case, we restricted ourselves to the study of run–time required for a WME action of a particular WMT, but now in the scope of a specific rule's RHS. Unfortunately, our measurements show that this time is also rather irregular. It highly depends on the number of actions performed in the network, that is to say on the number of matches and on the number of insertions and

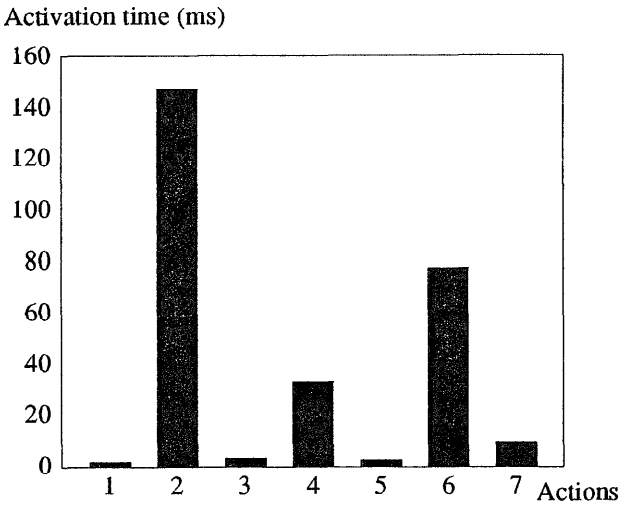

Figure 1: MAKE "Goal" action–time in rule "Sequence 5" in RUBIK

deletions. For example, in RUBIK[3] the rule "Sequence_5" has a very simple RHS, consisting only of one MAKE for the WMT "Goal". Figure 1 shows the evolution of the time required for the RHS firing of this rule. It was fired seven times during the execution for a specific cube configuration.

When applying the profiling method to the run–time of a rule's RHS, we encountered similar results. Each production system studied contained at least one rule showing irregular run–time behavior. Based on this fact we conclude that profiling of rule execution times is not adequate for run–time prediction.

To summarize, measurements of execution times for WME actions or RHSs cannot be used as a basis for run–time prediction. These discouraging results are quite general in nature. They hold for OPS–based as well as for PAMELA–based expert systems irrespective of their implementation method. We observed that for some WMTs the run–time behavior is sufficiently regular to perform prediction at this level of granularity. But we have also encountered WMTs for which it is not advisable to rely on rule– or WME action execution–time. In these cases more accurate tools based on data at a finer level of granularity are needed.

The Run–Time Prediction Method

As explained, simple profiling of run–time for RHSs and WME actions is not suitable for run–time prediction. Therefore, we investigate run–time estimations which rely on statistical data, dynamically gathered at node–level of the RETE network. The method presented in this section is designed to predict run–time for individual tokens originating from a WME action. More precisely, we are going to estimate run–time for the match phase when a plus–token enters the RETE network. We concentrate only on plus tokens because PAMELA treats minus–tokens in a very specific manner. The tokens in the network are linked together in a tree structure where each token points to its successors. Therefore, in contrast with the original RETE implementation, minus–tokens are not rematched. Instead, tokens are directly removed from token memories during the deletion phase[4]. Since the number of tokens being removed is known, we can calculate the exact run–time for minus–token treatment. Therefore, we concentrate only on plus–token treatment for the remainder of this paper.

Run–Time Estimation at Token Level

At the beginning of its flow through the RETE network, the token filters through one–input nodes. Only a small fraction of run–time is spent during one input–node treatment (Gupta 1986). Therefore, worst case run–time, which is linear in the number of incoming tokens for the one–input node network, is fairly small compared to overall match–time. This upper bound can be easily calculated by the assumption that no filtering takes place in one–input nodes. We assume an implementation in which tokens of one WME of a RHS are gathered at the end of one–input node chains. The one input–nodes are treated first and we then start the estimation task. Thus, we know the exact number of tokens entering the two–input node network.

The token flow in the network is characterized by several elementary operations. In a two–input node, each incoming token is compared against every token from the opposite memory. For each successful match, one new token is created and stored in the input memory of the successor. In order to estimate the time spent in a particular node during a plus–token treatment, we rely on the following numbers:

- the number of tokens entering this node,
- the time required for one token insertion into a token memory,
- the time required for a match.

All relevant token numbers for a specific plus–token (e.g. the number of new tokens produced by a two–input node) have to be estimated before actual matching starts. Because of the worst case combinatorial behavior of RETE and TREAT, it is not feasible to estimate these numbers only by evaluating the largest theoretical number of tokens that could flow through the network. Most of the time, this upper bound would be too large to be of any significance.

For the two–input nodes, our estimation is based on the implementation of a token flow ratio p_k in each two–input node (see Figure 2). The factor p_k represents the probability for a successful match at node k, when a token is compared against another. This probability factor p_k is updated after each performed plus–token treatment.

3. RUBIK is a production system with 71 rules which solves the Rubik's cube.

4. This is true for positive compare nodes only. Not nodes are treated separately.

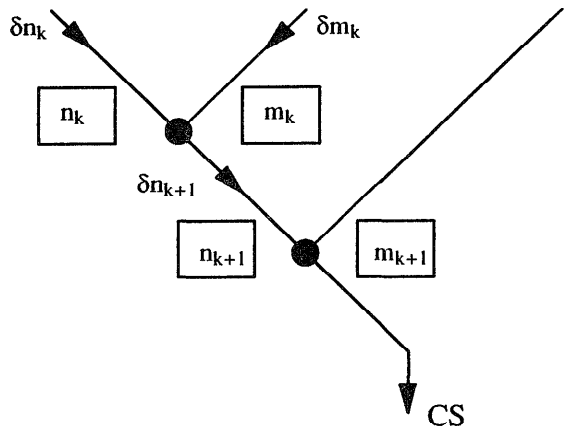

Figure 2: RETE network

For our estimation, we use the following numbers:

V_k ...	Number of matches at node k
m_k ...	Number of tokens in right memory of node k
n_k ...	Number of tokens in left memory of node k
δm_k ...	Number of plus tokens entering the right memory of node k
δn_k ...	Number of plus tokens entering the left memory of node k
p_k ...	Probability for a successful match at node k

For tokens entering the right memory of node k, we get

$$V_k = n_k * \delta m_k \quad (1)$$

$$\delta n_{k+1} = p_k * V_k \quad (2)$$

for a two–input node and

$$V_k = n_k * \delta m_k \quad (3)$$

$$\delta n_{k+1} = 0 \quad (4)$$

for a not node. For tokens entering the left memory of node k, we get

$$V_k = m_k * \delta n_k \quad (5)$$

$$\delta n_{k+1} = p_k * V_k \quad (6)$$

for a two–input node and

$$V_k = m_k * \delta n_k \quad (7)$$

$$\delta n_{k+1} = (1 - p_k)^{m_k} * \delta n_k \quad (8)$$

for a not node.

Note that equation (4) does not mean that no tokens are actually created in the RETE network. It only means that those possibly created tokens are negative tokens, and, therefore, we do not take them into account because of PAMELA's special minus–token treatment.

Using formulas (1) to (8), we can estimate the number of tokens that will be created in the network. In order to estimate the time required for the plus–token treatment, we measured the time for an insertion in a token memory and the time for a match on a T800 transputer[5] (Bahr et al. 1991), with a precision of 1 µs. We measured the following times:

- 5 µs for a match invoking a simple equality test, which is the most common situation. But since PAMELA offers the possibility to call a C–function within the test, the time for a single match may be significantly longer. We will thus also have to make our predictions with larger match–times.
- 50 µs for an insertion in a right token memory. This time does not depend on the node nor on the token whose length is always one in a right memory.
- (50 + 6*L) µs for an insertion in a left token memory, where L is the left token's length. (L actually means the number of WMEs it consists of). This dependence is due to the fact that PAMELA loops L times to copy the token into the memory.

Finally, the run–time prediction tool estimates the number of actions to be performed in the network (matches and insertions). In this way, match–time for the whole next WME action is estimated dynamically during run–time of the expert system.

The run–time prediction adds some burden to the whole execution–time of the expert system. However, compared to match–time this additional time is small. On our Transputer–based system, 50 µs run–time overhead must be taken into consideration per affected two–input node.

Results of the Run–Time Estimation Method for WME Actions

The measurements presented in this section were performed in the following way. Using the time required for each elementary action, we implemented a simulator, which computes the time that it would take on a T800 transputer. This method presents the great advantage that we can vary the values of the match–time and study their influence on the quality of our prediction.

Our measurements were performed on seven production systems:

- EMAB, the monkey and bananas production system written by NASA in an extended version with 29 rules
- RUBIK, a production system with 71 rules that solves the Rubik's Cube by J. Allen
- TOURNEY, a production system assigning match schedules to a bridge tournament with 16 rules
- WALTZ, a pattern recognition programme with 33 rules

5. Our parallel architecture for speeding–up Production Systems is based on T800 Transputers.

- VAHINE, a production system with 55 rules (by Alcatel – ELIN) that advises road maintenance in winter for given weather conditions.
- CARS, a production system with 62 rules (by Alcatel – ELIN) that dynamically maximizes traffic throughput on a network of roads and crossings.
- ALAMOS, an industrial version of a production system that performs alarm correlation on S12 which is Europe's largest public switching system. The expert system contains 112 rules at the moment and is a collaboration effort between Alcatel–ELIN in Vienna and BELL in Belgium.

VAHINE, ALAMOS and CARS are typical real–time production systems in the sense that they are continuously gathering data from the periphery via interrupts and polling mechanisms. CARS is especially time critical since it has to react immediately on traffic congestions. Although all three examples are only medium sized with respect to the number of rules they utilize vast amounts of data produced by the process periphery.

For reasons of statistical significance, we only present the results for those of the WMTs appearing sufficiently often during the expert system execution. The tables 1–7 display the averages and the variances of the ratio (real time)/(estimated time) for each of the seven applications.

WMT	monkey	goal	object
μ	0.99	1.02	1.03
σ	0.14	0.69	0.19

Table 1: EMAB

WMT	bot–cor	cub–ord	face	fr–face	face–rel	goal	sou	task
μ	1.00	1.03	1.05	1.30	1.00	1.90	0.98	1.00
σ	0.98	0.89	0.88	1.78	0.00	8.82	0.37	0.00

Table 2: RUBIK

WMT	apl	can	con	fou	pla
μ	1.00	1.00	191.24	1.09	1.00
σ	0.00	0.00	799.33	0.83	0.00

Table 3: TOURNEY

WMT	edge	junction	line	stage
μ	0.99	1.00	1.00	1.41
σ	0.20	0.00	0.00	1.49

Table 4: WALTZ

WMT	action	alarm	clock	message	sensor	weather
m	1.03	1.00	0.99	0.99	1.06	0.92
s	0.04	0.00	0.00	0.00	0.03	0.03

Table 5: VAHINE

WMT	time	car	cross
m	1.05	0.93	0.97
s	0.03	0.01	0.01

Table 6: CARS

WMT	switch	net_cou	erp_48	clock	net_eve	r_tun	r_port	alarm
μ	1.00	0.99	1.00	1.00	1.01	0.96	1.00	0.76
σ	0.00	0.00	0.00	0.00	0.01	0.02	0.00	0.06

Table 7: ALAMOS

These results were obtained using specific probability updating processes in two–input nodes. Let $\langle p_n \rangle$ be our estimated probability for time n and let p_n be the real ratio at time n (evaluated after match phase at time n) then we define:

$$\langle p_{n+1} \rangle = (3*p_n + \langle p_n \rangle)/4 \qquad \text{(ALGO 1)}$$

In the following discussion, we will refer to this updating procedure as ALGO 1. More obvious updating algorithms are ALGO 2 and ALGO 3 :

$$\langle p_{n+1} \rangle = (p_n + \langle p_n \rangle)/2 \qquad \text{(ALGO 2)}$$

$$\langle p_{n+1} \rangle = p_n \qquad \text{(ALGO 3)}$$

ALGO3 uses the real ratio p_n of the last RAC only. ALGO1 and ALGO2 also take the p_n's of all previous RACs into account, but they have different damping factors for older p_n's.

Altogether, the results in tables 1–7 show a fairly small variance, allowing us to expect in most cases an error lower than a factor 2. The factor is small compared to prediction based on simple profiling methods. The only exception to this favorable behavior is the WMT "Context" in TOURNEY. This WMT works like a goal switch during the execution of TOURNEY. Therefore, a MAKE statement of this type can reveal an unexpected behavior which invalidates the probabilities. In reference to OPS such a MAKE statement is called a context switch, and as yet we are unable to predict match–time for a such a context switch WME.

Nevertheless, the result is not as bad as it first seems to be. In fact, although those variances above give an indication for the accuracy of the prediction, Figure 3 provides a more relevant information. It shows the distribution of the ratio (real time)/(estimated time) for our three algorithms.

Figure 3 shows that about 50% of the time the ratio (real time)/(estimated time) is between 0.9 and 1.1 using the three algorithms. It also shows that, even in the worst case, 80% of the time the ratio real/estimated is between 0.1 and 10.0 and 60% of the time between 0.5 and 2.0. Therefore, and despite our large variance (table 3), the result is even there meaningful. Note also that the results are not significantly influenced by the updating algorithm we choose.

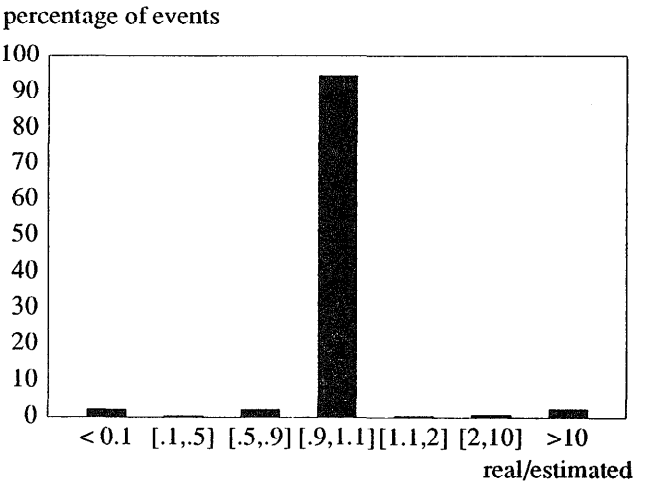

Figure 4: distribution of (real time)/(estimated time) for MAKE "goal" in RUBIK

Another interesting context switch is the "Goal" WME in RUBIK. The corresponding distribution of the ratio (real time)/(estimated time) has been calculated with ALGO1 and is shown in figure 4. Although the variance was once again not very small (table 2), a match–time with a precision better than 10% is predicted by the run–time estimation method more than 90% of the time.

In all other cases where a basic action does not correspond to a context switch, the run–time prediction is even more accurate. This accuracy is expressed by the very small variances of the corresponding WMTs in the tables 1–7.

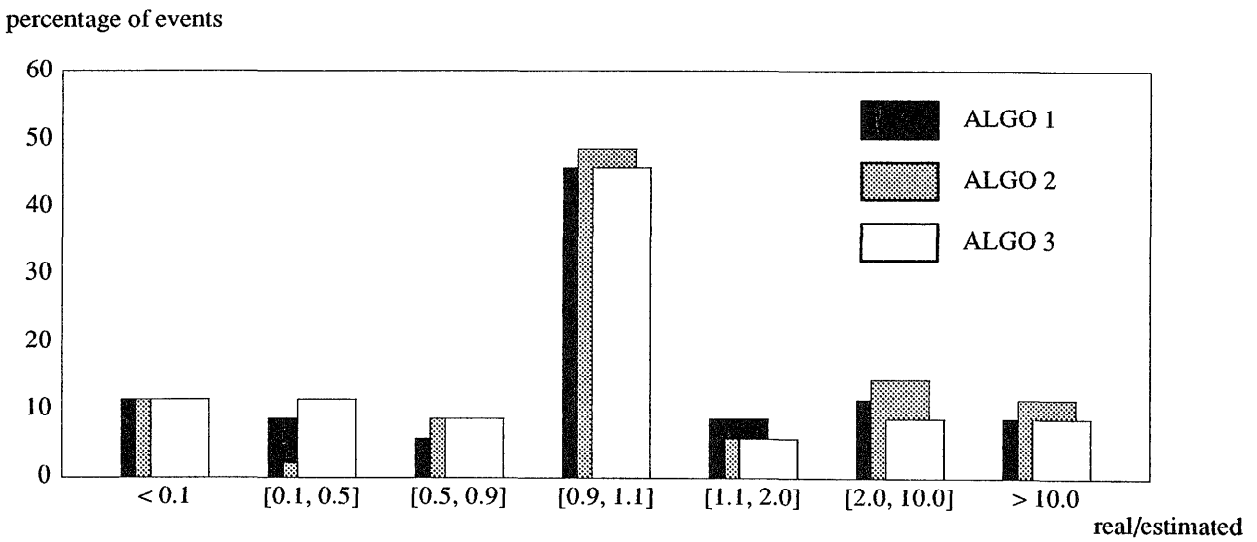

Figure 3: distribution of (real time)/(estimated time) for MAKE "Context" in TOURNEY

Results of the Run–Time Estimation Method for Rules

So far we are able to predict run–time for individual WME actions only. We want to go one step further and consider run–time prediction for a complete RHS, which may consist of several WME actions. The estimation mechanism is the same as for a single WME action but now the tokens of all WME actions belonging to one RHS are gathered at the end of the one–input node chains before the estimation task starts.

As a typical example we chose the RHS of the rule "minus_90" in RUBIK. The RHS consists of 21 CHANGE statements. Figure 5 shows the actual (simulated) run–time versus the estimated run–time. Although the estimated run– time deviates slightly from the actual one, the estimation models quite well the run–time behavior. The good quality of estimation could only be achieved by using our described fine granularity method. For all rules of the production systems presented in this paper we observed a notable correspondence between real and estimated rule execution times. Only in those cases where the RHS contains a context switch WME, the accuracy of the prediction is severely reduced.

Run–Time Estimation for a Complete Production System Task

The prediction of the total answering time for a complete task consisting of many reasoning cycles still remains an open problem. This is due to the fact that the sequence of rule firings cannot be known in advance. Even if we had tools that are able to extract relevant knowledge from the expert system's behavior we would have difficulties in predicting *exact* rule–firing sequences. Although some research has been done towards finding statical and dynamical rule dependences (Ischida 1990), a general solution for finding correct sequences is lacking. This is especially true for interrupt–driven production systems which are continuously gathering on–line data from industrial processes. Even if we knew rule–firing sequences in advance the reliability on the probabilities p_k and on the estimated number of tokens in the RETE memories decreases with increasing number of RACs to be predicted.

In intelligent high–level reasoning tasks the issue is not whether we can exactly predict run–time of specific tasks. The issue is more on the design of a problem–solving architecture that allows us to fall back on more primitive methods and exception handlers when time is running out.

Summary

In this paper we proposed a run–time estimation method that is able to predict match–time and rule execution time with significantly better precision than simple profiling methods. In the average case, we can predict run–time per rule and per WME action with an error of less than a factor of 2.

Although our method cannot give a rigorous upper bound for the run–time of WME actions and rules, it is well suited to systems that have soft time constraints. Such systems do not require guaranteed answering times for particular events but rather an estimation of the average time for a typical request. Based on these run–time estimations, exception handlers can be raised which are able to deliver incomplete or suboptimal results to a high level–reasoning system before a user–defined deadline is exceeded.

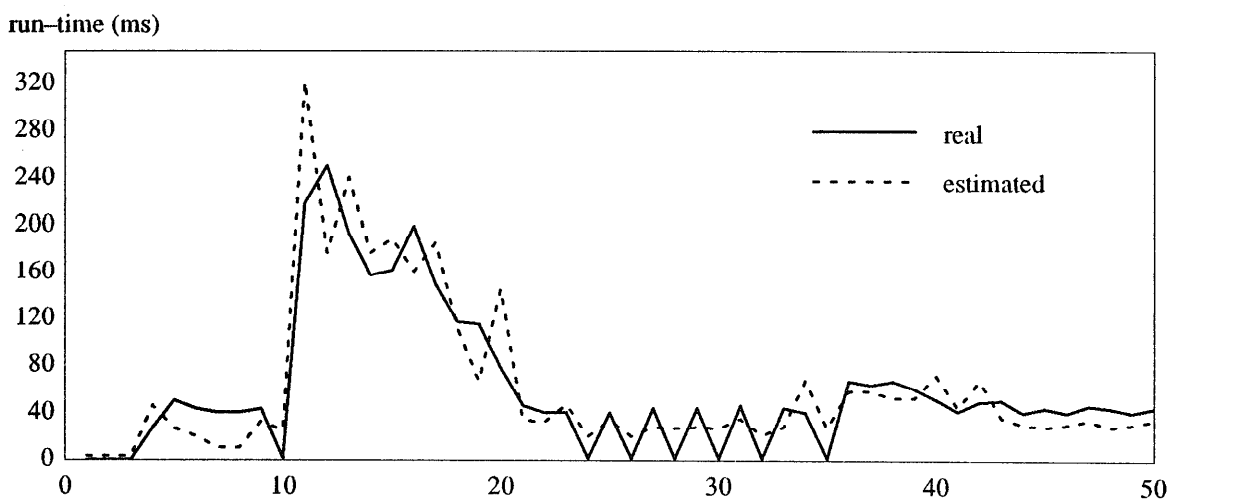

Figure 5 : Rule execution times versus estimated runtimes of rule "minus_90" in RUBIK

Acknowledgements

We owe thanks to Gilles Verteneul and Xavier Ursat who implemented the run–time prediction method for the PAMELA–C inference engine. This enabled us to perform predictability measurements and gave us a deeper insight into the statistical behavior of production systems. Furthermore we thank the AI group at the Alcatel–ELIN Research Center for providing us with production systems that are designed for application in industrial environments.

References

Acharya, A., Black, B., Paul, C.J., and Strosnider, J.K. 1991. Reducing Problem–Solving Variance To Improve Predictability. Communications of the ACM, Vol. 34: 81–93.

Bahr, E., Barachini, F., Doppelbauer, J., Gräbner, H., Kasparec, F., Mandl, T., and Mistelberger, H. 1991. A Parallel Production System Architecture. *Journal of Parallel and Distributed Computing* 13: 456–462.

Barachini, F. and Theuretzbacher, N. 1988. The Challenge of Real–Time Process Control for Production Systems. In Proceedings of AAAI–88, Vol. 2: 705–709.

Barachini, F. 1991. The Evolution of PAMELA. *Expert Systems, The International Journal of Knowledge Engineering*, Vol. 8: 87–98.

Brownston, L., Farrell, R. G., and Kant, E. 1986. *Programming Expert systems in OPS5*. Addison–Wesley.

Forgy, C. L. 1982. RETE : A Fast Algorithm for the Many Pattern/Many Object Pattern Matching Problem. *Artificial Intelligence*, Vol. 19: 17–37.

Forgy, C. L. 1981. OPS5 User's Manual, Technical Report CMU–CS–81–135. Carnegie–Mellon University.

Gupta, A. 1986. Parallelism in Production Systems. Ph.D. diss, Department of Computer Science, Carnegie–Mellon University.

Gupta, A., Forgy, C. L., Kalp, D., Newell, A., and Tambe M. 1986a. Results of Parallel Implementation of OPS5 on the Encore Multiprocessor, Technical Report. Department of Computer Science, Carnegie–Mellon University.

Gupta, A., Forgy, C. L., Newell, A., and Wedig, R. 1986b. Parallel Algorithms and Architectures for Rule–Based Systems. In *The 13th Annual International Symposium on Computer Architectures*, IEEE and ACM.

Gupta, A. and Tambe, M. 1988. Suitability of Message Passing Computers for Implementing Production Systems. In Proceedings of AAAI–88, Vol. 2: 687–692.

Haley, P. V. 1987. Real–Time for RETE. In Proceedings of ROBEX–87, Triangle Park, Instrument Society of America.

Ishida, T. 1990. Methods and Effectiveness of Parallel Rule Firing. In Proceedings of IEEE Conference on Artificial Intelligence Applications: 116–122.

Kalp, D., Tambe, M., Gupta, A., Forgy, C., Newell, A., Acharya, A., Milnes, B., and Swedlow, K. 1988. *Parallel OPS5 User's Manual*, Draft. Carnegie–Mellon University.

Laffey, T. J., Cox, P. A., Schmidt, J. L., Kao, S. M., and Read, J. Y. 1988. Real–Time Knowledge–Based Systems. In *AI Magazine*, Vol 9, no 1: 27–45.

Miranker, D. 1987. TREAT : A New and Efficient Match Algorithm for AI Production Systems. Ph. D. Thesis, Columbia University.

Tambe, M. and Newell, A. 1988. Some Chunks are Expensive. In Proceedings of the Fifth International Conference on Machine Learning: 451–458.

Tambe, M., Kalp, D., and Rosenbloom, P. 1991. Uni–Rete : Specializing the Rete Match Algorithm for the Unique–attribute Representation, Technical Report, CMU–CS–91–180. Carnegie–Mellon University.

Wang Chih–Kan, Mok Aloysius, K., Cheng Albert, M.K. 1990. MRL : A Real–Time Rule–Based Production System. In Proceedings of Real–Time Systems Symposium: 267–276.