TREE: the Heuristic Driven Join Strategy of a RETE-Like Matcher*

Jacques Bouaud
Departement Intelligence Artificielle et Medecine
INSERM U194 & Service d'Informatique Medicale AP-HP
91, boulevard de l'Hopital
F-75634 Paris Cedex 13 - FRANCE

Abstract

TREE is an optimized RETE-like pattern-matching algorithm. It has been designed for a production system whose restricted data formalism leads to a highly combinatorial join step like in SOAR. TREE aims at reducing the join search spaces without using hashing techniques. Its join strategy uses constraint propagation to define the solution space of a join, then a constraint relaxation to determine the index to be used in the join computation. Constraint relaxation is heuristic driven and based on the relational paradigm. Unlike RE IE, the indexing scheme TREE requires is not based on the membership of condition elements but on the sharing of references to symbols. On the basis of experimental evidence, TREE'S strategy showed better results than the standard RETE one. The number of comparisons during join steps has been reduced by a factor ranging from 1 to nearly two orders of magnitude.

1 Introduction

Match continues to be a problem for AI systems, especially for general purpose production systems like SOAR [Laird *et a*/., 1987]. In this kind of systems, a large flexibility in representations is provided through their decomposition into binary relations or into an object-attribute-value paradigm. Consequently, basic data elements are mainly restricted to triples. As pointed out by Tambe and Rosenbloom [1990], this kind of restricted formalism leads to a combinatorial match since a *condition element* (CE) of a rule can match nearly all the *working memory elements* (WMEs), and condition parts of rules contain many CEs. Several works attempt to improve match algorithms for such systems. With or without parallelism, efficiency gains can be obtained through condition ordering [Smith and Genesereth, 1985; Ishida, 1988; Miranker, 1990], by restricting expressiveness [Tambe *et a*/., 1990], or using hashing techniques [Gupta, 1987; Gupta *et a*/., 1988].

*This work has been partly supported by the Program for Artificial Intelligence (PRC-IA) of the French Ministry of Research.

TREE [Bouaud, 1992] is a state-saving match algorithm that has been designed for K, a production system with a restricted data formalism: WMEs are triples of symbols. TREE aims at reducing the size of join search spaces without using hashing techniques. It differs from RETE [Forgy, 1982] in its indexing scheme which does not rest on "memory support"[1] [McDermott *et* al., 1978], *t.e.* based on the CEs, but on the sharing of references to symbols. In this framework, we investigate a new join formulation. A constraint propagation and a heuristic relaxation based on the data paradigm is performed on the CEs in order to select the indexes to be used in the join step computation. These indexes are expected to be smaller than those built up from the CEs.

The paper is organized as follows: section 2 mentions RETE and join issues are revisited in the framework of the *"k-search model"* [Tambe *et* al., 1990]. The features of our system and the TREE algorithm are described in section 3. Section 4 reports experimental data about TREE'S strategy in comparison with RETE'S. These results and related works are discussed in section 5.

2 Background

We assume the reader is familiar with production system issues and the R.ETE algorithm. Condition parts of rules are mostly represented by conjunctions of CEs. Using the database terminology, an n-ary CE conjunction is represented by:

$$CE_{1\dots n} = CE_1 \bowtie CE_2 \bowtie \dots \bowtie CE_{n-1} \bowtie CE_n$$

where $\bowtie$ is the binary join operator. Tambe *et al.* [1990] showed that most match algorithms were based on the same model, referenced as the k-search model: an exhaustive standard backtracking search is made to exhibit every instantiation of an n-ary join. A conjunction is processed from left to right according to the recurrent formula: $CE_{1\dots k} = CE_{1\dots k-1} \bowtie CE_k$ until $CE_{1\dots n}$ is produced. The trace of this algorithm yields an instantiation tree where nodes at depth *k* are the partial instantiations corresponding to $CE_{1\dots k}$.

In state-saving algorithms, all intermediate join results are memorized in what we shall call by analogy with the *working memory* (WM) the *instantiation memory* (IM). Then, each partial instantiation is an *instantiation memory element* (IME). Problems of IM update arise when a new WME is added or an old one is removed.

Following the k-search framework, the problem of completing a given partial instantiation can be described as follows:

> Let i be an IME of $CE_{1...k-1}$ which is to be joined to CE_k. Computing $i \bowtie CE_k$ requires to find all the WMEs that both match CE_k and satisfy some compatibility tests with *i which* are mainly equality tests between variable values. These WMEs constitute the *solution space* of this join. We shall call such a join aiming at completing a given IME an *I-join.*

The problem of IM update when a WME is added is to find every IME that the WME completes. When a WME is deleted, all IMEs including the WME must be deleted. We shall address this problem later on. Adding a new WME can be described as follows:

> Let *w be* an added WME. For each CE* that matches w, the join $CE_{1...k-1} \bowtie w$ must be computed to update $CE_{1...k}$. This requires to find all the IMEs of $CE_{1...k-1}$ such that w satisfies their compatibility test. These IMEs constitute the *solution space* of this join. Similarly, such a join aiming at updating IM from a WME will bo called a *W-join.*

An *index* can be defined as a built-in relation: the set of its tuples can be directly enumerated. The indexing schemes of WM and IM are very important as indexes constitute practical *search spaces* for the joins. A necessary condition for an index being used in an I- or W-join is to include the solution space.

2.1 The RETE matcher

Like most matchers, RETE'S indexing scheme is based on the CEs of a rule base. The set of WMEs matching a given CE_k is stored in an *a-memory,* and the instantiations of a given $CE_{1...k-1}$ are stored in a *β- memory. β-*memories constitute the IM. RETE'S abstract procedures for IME and WME adds are given in Figure 1 where c, *p* and *r* are respectively the number of CE_k with an identical $CE_{1...k-1}$ (node sharing), of CEs matching w and of $CE_{1...k-1}$ whose CE_k matches *w.*

```
ADD-IME($i$ / $match(i, CE_{1...k-1})$)
    1.  Store $i$ in IM[$\beta(CE_{1...k-1})$]
    2.  Get $c$ $CE_k$
    3.  For each $CE_k$ do
        3.1 For each $w \in$ WM[$\alpha(CE_k)$] do
            If $test$ then ADD-IME($i \times w$)

ADD-WME($w$)
    1.  Select $p$ $CE/match(w, CE)$.
    2.  For each $CE$ do
        2.1.  Store $w$ in WM[$\alpha(CE)$]
        2.2.  Get $r$ $CE_{1...k-1}/CE_k = CE$.
        2.3.  For each $CE_{1...k-1}$ do
            2.3.1.  For each $i \in$ IM[$\beta(CE_{1...k-1})$] do
                If $test$ then ADD-IME($i \times w$)
```

Figure 1: RETE'S procedures for IME and WME adds.

α- and β-memories can be accessed from each other as they are connected in the RETE network that reflects the CE ordering. Data elements (WMEs) and partial instantiations (IMEs) are called *tokens* without distinction as they traverse identically the network in order to compute joins. Considering a linear RETE network, IMEs correspond approximately to left tokens, WMEs to right tokens, and I- and W-joins to left and right join activations. The .selection *step,* ADD-WME.l, deals with the maintenance of WM. A number of tests are applied to a WME for assigning it to the right a-memories. Unlike the join step, this step is not costly and can be optimized [Ghallab, 1981].

When memories are implemented as linear lists and as the basic binary join algorithm is the cross-product, ADD-IME is basically computed in $O(c.|CE_k|)$ and ADD-WME in $O(p.r.|CE_{1...k-1}|)$ where the notation $|X|$ stands for the cardinality of *X.* As for WME removal, RETE uses the same procedures as for adds except that store operations in memories are substituted by remove operations. Then, a WME removal is at least as costly than an add. This last point has often been criticized for state-saving algorithms [Miranker, 1990].

2.2 Improving RETE's joins: H-RETE

In practice and even with a good CE ordering, $|CE_k|$ and $|CE_{1...k-1}|$ can be large thus limiting RETE'S efficiency. Indexing techniques like hashing can be used to improve the join step by reducing the search spaces [Gupta, 1987]. Most of join tests being equality tests, memories can be hashed according to the variable to be tested. Two global hash-tables are used for WM and IM. The hash code for a token (WME or IME) is a function of the value of the variable to be tested and of the actual memory (α or β) location. As a result, hash-values constitute memory indexes for joins. Their sizes depend on the hash-table sizes and on the quality of the hash-function. Though the aim of hashing is to access the solution space, it only provides a good seardi space reduction, by a tenfold factor as usually acknowledged. But, it is also known that the overhead of computing hash-values can eliminate the benefits of this reduction for small sets compared to a list implementation [Aho *et* a/., 1983].

3 The TREE algorithm

3.1 Working framework

TREE has been implemented for a general purpose production system, K, where WMEs are triples of symbols, *e.g.* [F R V]. This structure is close to SOAR'S and the equivalent notation would be: *(triple identifier F attribute R value V).*

Indexation in K is not CE-based but organized on the criterion of sharing an identical reference to an identical symbol: WMEs are not aggregated according to their membership of CEs, but to the fact that they contain the same symbol at the same location (identifier, attribute, or value). For instance, the WME [F R V] is indexed 3 times: by F as first field, R as second, and V as third. In the following, we consider indexes as simple patterns. Hence, [F R V] is indexed by the patterns [F??], [?R?], and [??V], where ? are unnamed free variables.

The set of symbols defines the indexing scheme of K insofar as each symbol S in the system controls the indexes [S??], [?S?], and [??S] both in WM (for WMEs) and in IM (for IMEs). The six corresponding indexes are additional fields of a symbol and their content are simply implemented as linear lists. A symbol is unique and created when it is first referenced either at read time or during execution. As a result, symbols, and therefore their corresponding indexes, always exist before any WME that refers to them is created. When the WME [F R V] is created, the symbols F, R, and V are accessed and the WME is directly stored without test in the three indexes [F??], [?R?], and [??V]. This constitutes TREE's selection step which is independent of any rule base.

3.2 TREE's principles

TREE aims at reducing the size of join search spaces, making the most of K's actual indexing scheme. It mainly differs from RETE in the I-join formulation where contextual constraints (variable bindings) are used to determine the join index.

The problem of WME removal in our implementation is treated like in [Dixneuf *et al.*, 1988] where dependency links between IMEs are memorized. IMEs are organized in a tree structure according to the instantiation tree of the k-search algorithm. This tree is independent of the join strategy used to build or update it. When a WME is deleted, the subtrees the roots of which are the WME are deleted. Thus, WME removals are like the instantiation tree independent of the join strategy. We do not address here the evaluation of this method.

Now, let us consider the I-join $i \bowtie CE_k$ where $i \in CE_{1\dots k-1}$. Let CP_k^i be the *constrained pattern* obtained by the substitution of the bound variables of i in CE_k. The corresponding *constraint propagation* function is noted $\mathcal{P}$ and $CP_k^i = \mathcal{P}(i, CE_k)$. CP_k^i describes the solution space of the initial join: every present or future WME matching CP_k^i is successfully joined to i without test. For instance, let i be an instantiation where ?x is bound to A, CE_k = [?x P ?y], and the I-join be $i \bowtie$ [?x P ?y]. Applying $\mathcal{P}$ we obtain $\mathcal{P}(i,$ [?x P ?y]) = [A P ?y] and the initial join is equivalent to $i \times$ [A P ?y]. Now, two situations must be considered:

1. CP_k^i is an index: the search space WM[CP_k^i] is exactly the solution space and this I-join is computed optimally. For the future updates of i, i is stored in IM[CP_k^i]. The idea is to group IMEs according to the WMEs that will complete them, not to their membership of $CE_{1\dots k-1}$: every added WME matching CP_k^i completes i without test.

2. CP_k^i is not an index: some constraints on CP_k^i must be relaxed in order to find a subsuming *relaxed pattern* RP_k^i that is an index. We note the *constraint relaxation* function $\mathcal{R}$ and the initial join is turned into another one: $i \bowtie \mathcal{R}(i, CP_k^i)$. WM[$RP_k^i$] describes the new search space, wider than the solution space CP_k^i, but including it. WMEs matching RP_k^i complete i only if some compatibility test succeeds. For future updates, i is stored in IM[RP_k^i]. In our example, the constrained pattern [A P ?y] is not an index but [A ?t ?y] is, then $\mathcal{R}(i,$ [A P ?y]) could be [A ?t ?y] and therefore the initial join would be computed as $i \bowtie$ [A ?t ?y] with the test ?t = P.

With regard to the whole join algorithm, TREE computes the same joins as RETE, but initial I-joins $i \bowtie CE_k$ are reformulated as $i \bowtie \mathcal{R}(i, \mathcal{P}(i, CE_k))$, where $\mathcal{P}(i,p)$ returns the substitution of i in p, and $\mathcal{R}(i,p)$ returns an index subsuming p. In the following, every pattern p is considered within an I-join (i given) so that we use the notation $\mathcal{R}(\mathcal{P}(p))$ in place of $\mathcal{R}(i, \mathcal{P}(i,p))$. RETE's procedures are consequently slightly modified for TREE (Figure 2). ADD-IME is basically computed in $O(c.|\mathcal{R}(\mathcal{P}(CE_k))|)$ and ADD-WME in $O(3.|\text{IM}[Index]|)$.

```
ADD-IME($i$/ $match(i, CE_{1\dots k-1})$)
    1.  Get $c$ $CE_k$
    2.  For each $CE_k$ do
        2.1.  Let $Index = \mathcal{R}(i, \mathcal{P}(i, CE_k))$
        2.2.  Store $i$ in IM[$Index$]
        2.3.  For each $w \in$ WM[$Index$] do
                If $test$ then ADD-IME($i \times w$)

ADD-WME($w$/$w$ = [F R V])
    1.  For each $Index \in$ {[F??],[?R?],[??V]} do
        1.1 Store $w$ in WM[$Index$]
        1.2 For each $i \in$ IM[$Index$] do
                If $test$ then ADD-IME($i \times w$)
```

Figure 2: RETE's modified procedures for TREE.

Since only the search spaces of joins are changed, it is not possible to demonstrate which of TREE and RETE performs better joins without explicit relations between index cardinalities, *i.e.* $|CE|$ vs. $|\mathcal{R}(\mathcal{P}(CE))|$, and $|CE_{1\dots k}|$ vs. $|\text{IM}[index]|$. If $\mathcal{P}$ could be used alone one could not perform a better join computation.[1] But, building an architecture providing all possible indexes is unreasonable. $\mathcal{R}$ could be also easily directed toward the initial CE, but we would return to the RETE model. As a consequence, efficiency depends on $\mathcal{R}$ which itself depends on the predefined indexing scheme.

3.3 A heuristic constraint relaxation

In our implementation, $\mathcal{R}$ is heuristic based. Unlike $\mathcal{P}$ which depends on both a CE and its local constraints (variable bindings), $\mathcal{R}$ only considers its argument, a pattern, to deliver an index. The heuristic principles rest on the expected cardinalities of different pattern types which themselves depend on the properties of the formalism the WMEs are used for. Since triples are mainly used in the relational paradigm, we assume that the notation [F R V] expresses that F is in relation R with V. We also assume that the number of relations is relatively small compared to the number of other symbols. In the general case and back to pattern cardinalities, $|[?R?]|$ is expected to be much greater than $|[F??]|$ and $|[??V]|$, and, $|[??V]|$ to be close to, or even smaller than, $|[F??]|$, whatever the symbols F, V, and the relation R.

TREE's I-join formulation is realized at compile-time. Within an ordered conjunction, $\mathcal{R}o\mathcal{P}$ is virtually applied

[1] This is only true considering a RETE-like algorithm, with a given initial WM and a given conjunction ordering.

to each CE. If the type of *V(CE)* is [F??], [?R?] or [??V], where letters correspond to known places (symbols or bound variables), then it corresponds to an index that the matcher will use. In the other situations, *V(CE)* is not an index and constraint relaxation applies. Here is how R selects a subsuming index for each of the five non index pattern types according to our assumptions:

- $\mathcal{R}$([?RV]) = [??V], instead of [?R?];
- $\mathcal{R}$([FR?]) = [F??], instead of [?R?];
- $\mathcal{R}$([F?V]) = [??V], instead of [F??];
- $\mathcal{R}$([FRV]) = [??V], instead of [F??] or [?R?];
- [???], which describes the whole WM, is a special case. We suppose that such a CE will never remain unconstrained ($\mathcal{P}$([???]) $\neq$ [???]).

The plain lines in Figure 3 represent our constraint relaxation function between pattern types[2] whereas dashed lines represent the subsumption relation and illustrate how constraint propagation can apply.

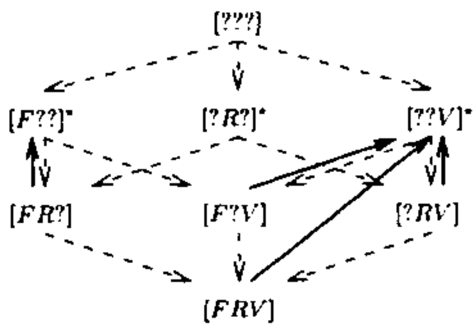

Figure 3: Subsumption links and constraint relaxation.

Now the question is: are the search spaces of I-joins and W-joins defined by RoP applied to the CEs smaller than those defined by the CEs themselves?

4 Experimental measurements

In order to evaluate TREE'S join strategy and compare it with a CE-based join, some experimental measurements were carried out with K. Five rule-based programs developped in our laboratory were used as benchmarks: MAB, the "monkey and bananas" problem [Brownston *et al.*, 1985]; ALEXIA, a qualitative model for hypertension [Bichindaritz and Seroussi, 1992]; CHART, a syntactic chart parser; AMD, a semantic analyzer for natural language [Cavazza and Zweigenbaum, 1992]; and ABACAB, a blackboard controller [Bachimont, 1991]. Table 1 provides some of their external characteristics.

4.1 Pattern cardinality measurements

We first studied the cardinality in WM of every possible pattern at the end of each execution. The average cardinalities of each pattern type are reported in Table 2. As these numbers correspond to the cardinalities of potential CEs, search and solution spaces for I-joins, the CE

[2]Pattern types with a * correspond to indexes.

Characteristics	MAB	ALEXIA	CHART	AMD	ABACAB
# of rules	26	15	95	196	107
# of distinct CEs	26	35	78	197	171
mean # of CEs/rule	7.3	8.4	4.1	8.2	8
Initial # of WMEs	67	3,734	97	1,610	1,028
Final # of WMEs	157	3,772	1,868	1,776	2,302
# of WM changes	267	274	1,771	176	2,088
% of adds	80%	54%	100%	97%	91%
# of firings	43	128	1 423	100	3 289

Table 1: Characteristics of programs and executions.

Pattern	MAB	ALEXIA	CHART	AMD	ABACAB
[???]	157	3,772	1,868	1,776	2,302
[?R?]*	11.21	87.72	51.89	24.32	14.97
[F??]*	2.96	11.86	4.21	2.77	5.85
[??V]*	2.24	1.71	1.91	2.01	2.83
[?RV]	1.96	1.57	1.85	1.75	1.72
[FR?]	1.00	1.66	1.30	1.23	1.41
[F?V]	1.00	1.03	1.00	1.03	1.07
[FRV]	1	1	1	1	1

Table 2: Average cardinality of each pattern type.

tranformations $\mathcal{P}$ and $\mathcal{R}$ can be quantified. From now on, it must be noted that all quantitative information is given on average.

As expected, the cardinality of [?R?] is always the highest ([???] excepted). $\|[?R?]\|$ is much greater than the cardinalities of the other indexes, [F??] and [??V], by a factor ranging from 2.6 ($\|[?R?]\|/\|[F??]\|$, ABACAB) to 51 ($\|[?R?]\|/\|[??V]\|$, ALEXIA). $\|[??V]\|$ is only slightly smaller than $\|[F??]\|$, the reduction factor is not very important, no more than 7 ($\|[F??]\|/\|[??V]\|$, ALEXIA). This ensures that the heuristic foundations for TREE were correct at least for these programs.

Considering the constraint propagation function $\mathcal{P}$ and a CE, we know that $|\mathcal{P}(CE)| \leq |CE|$. Thus, the reduction or gain factor $g = \frac{|CE|}{|\mathcal{P}(CE)|}$ due to $\mathcal{P}$, if applied in some relevant cases, could vary from 1 (no constraint) to more than 1000 ($\|[???]\|/\|[FRV]\| = 3772$, ALEXIA). Now, let us consider the constraint relaxation function $\mathcal{R}$ for non index patterns. The cardinalities of [F??] and [??V], which are the indexes selected by $\mathcal{R}$ in our case, are rather low, 12 being a maximum ([F??], ALEXIA). As the cardinalities of non index patterns are yet lower, the expansion or loss factor $l = \frac{|\mathcal{R}(\mathcal{P}(CE))|}{|\mathcal{P}(CE)|}$ due to $\mathcal{R}$ ranges from 1 ($\|[??V]\|/\|[?RV]\|$, CHART) to 7 ($\|[F??]\|/\|[FR?]\|$, ALEXIA). Thus, we can see that for I-joins, the gain factor due to $\mathcal{P}$ could reach three orders of magnitude whereas the loss factor due to $\mathcal{R}$ would be always inferior to one order of magnitude.

4.2 Analysis of pattern distributions

In a second step, the rule bases were analyzed to study qualitatively how $\mathcal{P}$ and $\mathcal{R}$ apply. Table 3 reports the pattern type distributions of: the CEs, the CEs constrained by $\mathcal{P}$, and the indexes resulting from $\mathcal{R}\circ\mathcal{P}$ applied to the CEs. A CE column gives the CE distribution in the rule base. It also corresponds to the kind of search spaces RETE would use for I-joins. A $\mathcal{P}$ column describes

Pattern	Mab			Alexia			Chart			Amd			Abacab		
	CE	$\mathcal{P}$	$\mathcal{R}\circ\mathcal{P}$	*CE*	$\mathcal{P}$	$\mathcal{R}\circ\mathcal{P}$	*CE*	$\mathcal{P}$	$\mathcal{R}\circ\mathcal{P}$	*CE*	$\mathcal{P}$	$\mathcal{R}\circ\mathcal{P}$	*CE*	$\mathcal{P}$	$\mathcal{R}\circ\mathcal{P}$
[???]	-	–		–	–		–	–		99	–		8	–	
[?R?]*	96	7	7	42	–	–	297	1	1	444	53	53	457	15	15
[F??]*	–	–	53		-	51	-	-	296	–	26	304	–	–	318
[??V]*	–	–	129	–	–	75	–	–	93	8	6	632	–	–	471
[?RV]	93	64		58	30		93	93		408	235		266	146	
[FR?]	–	53		10	51		–	296		30	278		56	318	
[F?V]	–	–		–	–		-	–		–	59		–	–	
[FRV]	–	65		16	45		–	–		–	332		17	325	

Table 3: CEs, constrained CEs and TREE'S constrained/relaxed CEs.

the solution spaces of these I-joins, and a R*OP* column, their search spaces with TREE.

Comparing column CE with Ro'P illustrates how the initial search spaces defined by the CEs are transformed by TREE. With the numbers of Table 2 in mind, one should notice that for each program:

1. Most of the CEs correspond to high cardinality patterns, [?R?] being the most frequent and [???] sometimes used (AMD and ABACAB);
2. Most of the CEs are constrained, especially [?R?] and every [???], and the constrained CEs correspond to low cardinality patterns;
3. As most of the constrained CEs are non index patterns, R is often applied and [F??] and [??V] are mainly used, the cardinalities of which are not high.

These points highlight that high gain factors *g* due to *V* and low loss factors / due to *7Z* are often expected, resulting in a combined reduction ratio $\frac{|CE|}{|\mathcal{R}(\mathcal{P}(CE))|} = \frac{g}{l}$ due to Ro*V* greater than 1. However, this would be only valid for I-joins, but in practice the previous arguments are not sufficient to determine whether the loss due to R discards the benefits from *V* or not. Yet, no similar information is available for W-joins.

4.3 Program execution measurements

Finally, each program has been run with three different join strategies. The only difference between them lies in the indexes used to search WMEs (in I-joins) and to store IMEs (for W-joins): TREE uses the indexes determined by the heuristic *TZoV* function applied to the CEs. RETE uses the initial CEs as indexes and then simulates a standard RETE linear network. OPTI is a simulation; only *V* is applied. It is as if the solution space of every join was directly accessible: search spaces were exactly solution spaces.

The number of comparisons per join is used as the metric for comparing the strategies as it corresponds to the size of the search spaces explored during join attempts. The smaller this number, the more efficient the join strategy. Moreover, this metric is adequate to compare indexing schemes [Miranker *et* al., 1990]. For a given program, identical initial WM and condition ordering were used. Comparing the number of generated "tokens" (IMEs) as in [Nayak *et* a/., 1988] is inappropriate here because the general algorithm is the same, used in the same conditions, and consequently produces identical IMEs: every join has the same solution space whatever the strategy. As every local join is optimal in OPTI, this "virtual" strategy is the best one and neither TREE nor RETE can outperform it: in the same conditions, it is not possible to do less comparisons.

Statistics about the number of comparisons per I- and W-joins are recorded in Table 4. Only non empty join attempts, *i.e.* with at least one comparison, were considered. Runtime performance is not significant here since the RETE and OPTI strategies were emulated from TREE'S. Although each result could be separately discussed, some points must be emphasized.

1 - The average number of comparisons per join *(Avg.)* is lower with TREE than with RETE whatever the program. Thus, actual search spaces defined by Ro*V,* for both I- and W-joins, are smaller than those defined by the CEs. The resulting reduction ratio ranges from 1 up to nearly two orders of magnitude and the same trend is also observed for standard deviation *(SD)* and maxima *(Max.).*[3]

2 - For every program, MAB excepted, TREE'S number of non empty join attempts is approximately the same as, or even smaller than RETE'S. This being combined with the previous result explains that the total number of comparisons per execution *(Sum)* is always much smaller with TREE than with RETE by a factor ranging from 10 (W-joins of AMD) to 74 (W-joins of ABACAB).

3 - MAB, which is a simpler program, is an interesting case for it illustrates a situation where RETE performs better than TREE: RETE'S total number of comparisons for W-joins is lower and though TREE'S search spaces are smaller than RETE'S, more join attempts are performed, just enough to cancel out the previous benefits.

4 - The number of comparisons of OPTI corresponds to the number of solutions of the joins. With this number and those of RETE and TREE, a global gain factor *G* due to constraint propagation and a global loss factor *L* due to constraint relaxation can be estimated. Table 5 gives such ratios. *G* is RETE/OPTI and ranges from 5 to 750 whereas L, which is TREE/OPTI, never exceeds 22. As previously noted the global resulting reduction ratio RETE/TREE is between 1 and nearly two orders of magnitude. Actually, this shows that in our most complex programs the effect of *V* outdoes the effect of R even during execution, then producing an actual search space reduction. MAB excepted, this reduction seems to increase with the number of WM modifications. As

[3] These properties are very interesting in the framework of parallel implementations [Gupta, 1987].

Program	Method	# of I-joins	# of comparisons/I-join [Sum	Avg.	(SD	Max.)]	# of W-joins	# of comparisons/W-join [Sum	Avg.	(SD	Max.)]
MAB	OPTI	413	413	1.0	0.0	1	138	496	3.6	2.9	10
	TREE	763	2,480	3.2	2.1	11	201	2,744	13.6	26.6	150
	RETE	721	2,906	4.0	3.9	20	151	2,511	16.6	30.3	150
ALEXIA	OPTI	2,006	4,253	2.1	5.9	114	57	103	1.1	2.9	22
	TREE	2,752	17,803	6.5	5.7	115	57	142	2.7	2.7	22
	RETE	2,752	305,933	111.2	81.2	263	102	3,929	38.5	40.1	134
CHART	OPTI	1,975	1,975	1.0	0.0	1	1,333	2,075	6.6	1.4	13
	TREE	1,975	8,018	4.1	0.7	9	2,086	8,617	4.1	8.4	64
	RETE	2,774	422,125	152.2	108.8	496	2,289	293,619	128.3	243.2	1,149
AMD	OPTI	3,511	13,159	3.7	8.7	48	204	1,352	6.6	10.0	47
	TREE	14,444	292,757	20.3	31.1	101	394	15,800	40.1	168.5	1,489
	RETE	14,558	3,451,961	237.1	602.1	2,102	540	170,228	315.2	355.1	4,350
ABACAB	OPTI	4,044	6,367	1.6	2.4	28	1,024	2,921	2.8	4.1	44
	TREE	7,258	56,608	7.8	6.0	37	2,195	29,442	13.4	36.4	612
	RETE	7,194	3,257,925	452.9	777.5	2,307	3,993	2,189,330	548.3	533.6	1,835

Table 4: Statistics of the # of comparisons per non empty join attempt.

Ratio	Join type	MAB	ALEXIA	CHART	AMD	ABACAB
RETE/OPTI	I-joins	7.0	71.9	213.7	262.3	511.7
	W-joins	5.1	38.1	141.5	125.9	749.5
TREE/OPTI	I-joins	6.0	4.2	4.1	22.2	8.9
	W-joins	5.5	1.4	4.1	11.7	10.1
RETE/TREE	I-joins	1.2	17.2	52.6	11.8	57.5
	W-joins	0.9	27.7	34.1	10.8	74.4

Table 5: Reduction ratios of the total number of comparison due to the different join strategies.

a global result, TREE does less comparisons than RETE for both I- and W-joins.

5 Discussion and related works

On the basis of experimental evidence, TREE has shown a better performance than a standard RETE on the overall number of comparisons performed during join attempts. This result has been obtained with five programs in different areas that all satisfied the assumption of the relational paradigm. To explain this result, we can observe that most of the CEs mention explicit relations: [?x R ?y]. These CEs are also nearly always linked through variable sharing such that often ?x or ?y are bound. In this situation, TREE choses [F??] or [??V] as indexes. For the join step, TREE prefers using the *contextual constraints* of variables bindings to focus the searches instead of the *definitional constraints* of the CEs that RETE would use. Under our assumption and with the previous results, it appears that the contextual constraints are stronger than the definitional ones. Moreover, this tendancy is not expected to collapse as the WM size increases. In this situation, we expect new symbols to be created, new connections to be made, but no (or few) new relations to appeal*. As a result, |[?/??]| would increase whereas both |[F??]| and |[??V']| would very likely remain constant. Therefore, the global reduction ratio due to TREE is also expected to increase making still more difference with RETE. This is an hypothesis that would have to be studied and confirmed.

Several other works attempt to improve the join step by the use of indexing techniques which are not (only) based on CEs. The domain-based indexing of a new version of TREAT [Miranker *et al.*, 1990] relies on the actual values of variables in a rule.

The role of hashing in H-RETE is also to provide smaller join search spaces. It is usually acknowledged that hashing techniques provide a tenfold reduction. Gupta *et al.* [1988] exhibit on three OPS5 programs search space reductions ranging from 1 to 12. This numbers indicate that H-RETE and TREE provide the same range of search space reduction though we did not test the H-RETE strategy on our programs. Nevertheless, the overhead due to hash-code computations can be important and can sometimes eliminate the benefits of hashing. Compared to that, TREE'S indexing scheme maintenance cost is low even compared to RETE'S because no test is performed.

The predefined indexing scheme provides some additional advantages. Random WM access is handled efficiently either by static pattern-matching or by program. Since it is independent of the CEs and therefore of any rule base, new rules can be dynamically compiled without any WM re-structuration. As a RETE-like algorithm, TREE also handles some features that have not been discussed such as "node sharing", explicit right join operator, and negated CE conjunction.

Though TREE and H-RETE would provide quantitatively identical effects, we would like to point out some qualitative difference. The nature of an index is symbolic with TREE versus numeric with H-RETE and the index choice is made according to a heuristic strategy versus to some hashing algorithm. The index choice can be made at compile time with TREE whereas it must be made at runtime with H-RETE. Lastly, changing the index choice can be made according to some qualitative reasonning versus some numerical algorithm. This allows us to consider the possibility of taking into account some properties of representations in order to improve production system performance.

6 Conclusion

In the framework of K, TREE'S heuristic join strategy has been shown as a challenger for RETE. We are convinced that TREE can be applied to other systems with a restricted WM formalism because CEs are often weakly selective but often constrained. For instance, we believe TREE would certainly perform efficiently for SOAR as it fulfills these conditions and satisfies the assumption of relational paradigm. Though we did not test it, TREE would probably not fit for OPS5 programs as their CEs are likely more discriminant than symbol sharing.

However, TREE can still be optimized. It could be interesting to explore other constraint relaxation strategies that would take into account the CEs and the representation framework. A join strategy exploiting a combination of TREE'S and RETE's indexing schemes would certainly lead to better results than TREE alone. This has to be tested in the future.

Acknowledgements

We would like to thank B. Bachimont, I. Bichindaritz, M. Cavazza, and L. Dore for gracefully providing the rule-based programs we used.

References

[AAAI, 1988] Proceedings of the 7 ih National Conference on Artificial Intelligence, St. Paul, MN, August 1988. AAAI.

[Aho et ai, 1983] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. Data structures and algorithms. Addison Wesley, 1983.

[Bachimont, 1991] B. Bachimont. A logical framework to manage coherence and convergence in blackboard architectures: a proposal. In Proceedings of the A AAI' 91 Workshop on Blackboard Systems, Anaheim, CA, 1991. AAAI.

[Bichindaritz and Seroussi, 1992] I. Bichindaritz and B. Seroussi. Contraindre l'analogie par la causalite. Technigue et Science Informatiques, 11(4):69 98, 1992.

[Bouaud, 1992] J. Bouaud. TREE : une strategie de jointure heuristique pour algorithme de filtrage. Revue dIntelligence Artificielle, 6(4):457-493, 1992.

[Brownston et al, 1985] L. Brownston, R. Farell, E. Kant, and N. Martin. Programming expert systems in OPS5. An introduction to rule-based programming. Addison Wesley, 1985.

[Cavazza and Zweigenbaum, 1992] M. Cavazza and P. Zweigenbaum. Extracting implicit information from free text technical reports. Information Processing and Management, 28(5):609-618, 1992.

[Dixneuf et ai, 1988] P. Dixneuf, A. Meller, and M. Porcheron. ELOISE's heart, an efficient frame for production system execution. In Proceedings of the 8 th European Conference on Artificial Intelligence, pages 24-26, Munich, FRG, August 1988. ECAI.

[Forgy, 1982] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, 19:17-37, 1982.

[Ghallab, 198l] M. Ghallab. Decision trees for optimizing pattern-matching algorithms in production systems. In Proceedings of the 7 th International Joint Conference on Artificial Intelligence, pages 310-312, Vancouver, Canada, August 1981. IJCAI.

[Gupta et ai, 1988] A. Gupta, M. Tambe, D. Kalp, C. Forgy, and A. Newell. Parallel implementation of OPS5 on the Encore multiprocessor: results and analysis. International Journal of Parallel Programming, 17(2):95-124, 1988.

[Gupta, 1987] A. Gupta. Parallelism in Production Systems. Pitman; Morgan Kaufmann Publishers, 1987.

[Ishida, 1988] T. Ishida. Optimizing rules in production system programs. In AAAI [1988J, pages 699704.

[Laird et al., 1987] J.E. Laird, A. Newell, and P.S. Rosenbloom. SOAR: An architecture for general intelligence. Artificial Intelligence, 33:1-65, 1987.

[McDermott et al., 1978] J. McDermott, A. Newell, and J. Moore. The efficiency of Certain Production System Implementations, pages 155-176. Academic-Press, New York, 1978.

[Miranker et al, 1990] D. P. Miranker, B. J. Lofaso, G. J. Farmer, A. Chandra, and D. A. Brant. On a TREAT-based production system compiler. In Proceedings of the 10 ih International Workshop Expert Systems & their Applications, pages 617-630, Avignon, France, May-June 1990. EC2.

[Miranker, 1990] D. P. Miranker. TREAT: a New and Efficient Match Algorithm for AI Production Systems. Pitman; Morgan Kaufmann Publishers, 1990.

[Nayak et al., 1988] P. Nayak, A. Gupta, and P. Rosenbloom. Comparison of the RETE and TREAT production matchers for Soar (a summary). In AAAI [1988], pages 693-698.

[Smith and Genesereth, 1985] D. E. Smith and M. R. Genesereth. Ordering conjunctive queries. Artificial Intelligence, 26:171-215, 1985.

[Tambe and Rosenbloom, 1990] M. Tambe and P. S. Rosenbloom. A framework for investigating production system formulations with polynomially bounded match. In Proceedings of the 8 th National Conference on Artificial Intelligence, pages 693-700, Boston, MA, July-August 1990. AAAI.

[Tambe et ai, 1990] M. Tambe, A. Newell, and P. S. Rosenbloom. The problem of expensive chunks and its solution by restricting expressiveness. Machine Learning, 5(3):299-348, 1990.