Average case complexity analysis
of RETE pattern-match algorithm
and average size of join in Databases

Luc ALBERT *

Abstract. The RETE algorithm [Forg 82] is a very efficient method for comparing a large collection of patterns with a large collection of objects. It is widely used in rule-based expert systems. We studied ([AF 88] or [Alb 88]) the average case complexity of the RETE algorithm on collections of patterns and objects with a random tree structure. Objects and patterns are often made up of a head-symbol and a list of variable or constant arguments (OPSV [Forg 81], Xrete [LCR 88] ...). In this paper, we analyse the theoretical performance of RETE algorithm on this widely used type of pattern and object with the theory of generating functions. We extend this work to the study of the performance of composed queries in relational Databases and we generalize Rosenthal’s theorem on the average size of an equijoin [Rosen 81]. We give some numerical examples based on our results.

CR classification : F.2 (Analysis of algorithms), I.2 (Artificial Intelligence), H.2 (Database).

I. Introduction

The RETE pattern match algorithm [Forg 79] [Forg 82] has been introduced by C. Forgy in the line of his work on production systems [Forg 81].

Production systems, or more generally rule-based systems, are widely used in Artificial Intelligence for modelling intelligent behaviour [LNR 87] and building expert systems. They are quite easy to use and have many advantages: modularity, relative independence of each rule and the same expressivity as a Turing machine. However the inference engine is algorithmically inefficient. The most time consuming process in a rule-based system is the pattern match phase that consists of maintaining the set of satisfied rules among changes in the data base. This computation can represent more than 90 % of the overall computation time in an application [DNM 78].

RETE algorithm is an efficient method for computing the set of satisfied rules incrementally after each rule execution. The incremental computation is justified in expert systems applications by the fact that the execution of a rule affects a relatively small number of objects (or facts or terms) in comparison to the total number of objects. Therefore most of the previous pattern match work remains valid. RETE algorithm realizes a total indexing of the data base according to rule conditions. Conditions common to several rules are shared in such a way that several rules can be found to be satisfied by testing some patterns only once.

Forgy [Forg 79] proved, thanks to simplifying hypotheses, that with RETE algorithm the worst case time complexity for computing the set of satisfied rules is linear in the number of rules, and polynomial in the number of objects (with degree being the maximum number of conditions in a rule). In the best case the complexity is a constant. Between these extremes the sensitivity of pattern match time to the size of the data base is highly dependent on rule characteristics. We already studied the theoretical average case complexity of RETE algorithm when it compares objects and patterns having any tree structure. ([AF 88], [Alb 88]). In real applications, objects and patterns are often made up of a function symbol and a list of variable or constant arguments. The height of their usual tree structure is therefore one and we analyse in this article the average performance of the algorithm in the case of these "flat terms".

* Institut National de Recherche en Informatique et Automatique, Domaine de Voluceau, Rocquencourt, BP 105, 78150 Le Chesnay Cedex France. mail : albert@inria.inria.fr

Laboratoire Central de Recherches, Thomson-CSF, Domaine de Corbeville, BP 10, 91401 Orsay Cedex France.
There are many motivations in searching for a more accurate model of computation and an average case complexity analysis of RETE algorithm. First, RETE algorithm admits many variants and optimizations, concerning the representation of local memories [Forg 79], the sharing of conditions (the ARBRE D'UNIFICATION : [Gha 87], [Alb 88]), the computation of joins [Mir 87], the total compilation principle [Fag 86] [Fag 88], the parallelization of the algorithm [Gupt 84] etc ... An average case complexity analysis can be used to evaluate these optimizations and propose new ones. Second, run-time performance prediction is a necessity for the development of real-time expert systems [WGF 86], [SF 88]. A mathematical model of run-time requirements can be used to extrapolate from the run-time performance of a prototype the performances of the expert system in real size. It define the range of its applicability in terms of number of objects that can be treated at a given time. Third, a mathematical model can be used also to work out significant benchmarks in order to compare several implementations according to the relevant characteristic parameters of a knowledge base [GF 83]. Lastly, we shall show that all this analysis applies not only to the study of the performance of pattern-match in expert systems but also to the estimation of the average size of composed queries in Relational Databases. In this paper we present an average case complexity analysis of RETE algorithm and of the average size of composed queries in Relational Databases using the generating function theory ([Fla 85], [FS 86], [FV 87]) (that introduces mathematical methods of independent interest). One can find a more detailed version of this study in [Alb 89].

Only equality tests are considered first. In Part 2, we briefly present RETE algorithm, and develop an Example. We define also the cost of this algorithm and precise the fundamental quantities for its computation. In Part 3, we introduce the generating function theory that is used to analyse the average case complexity of algorithms (3.2) and the asymptotic analysis necessary to simplify the expressions we previously obtained (3.3). Thus we obtain a result under a first model in which the Database is represented as an ordered list of terms (3.4). In Part 4, we determine the complexity under a second model that considers the Database as a multiset of terms. In Part 5, we extend the previous results by considering several separate ranges of variation for constants (5.1). Then we generalize our results for taking into account different frequency coefficients for symbols (5.2) (by the way we shall consider inequality tests). In Part 6, we consider the negation between arguments and between patterns.

Lastly in Part 7, we apply all these results to the study of composed join in Relational Database. We illustrate the results we obtain throughout the article with some examples. The example of figure 1 is numerically developed in the appendix.

II. RETE algorithm

2.1. Presentation

The production systems we shall consider are composed of a fixed set, denoted by RB (for Rule Base), of if-then rules called productions, and a changing set of facts, called the Working Memory and denoted by WM. Facts are formed on a finite alphabet F of function symbols given with their arity. Arguments are taken in a finite set C of symbols of arity 0, the constants. For instance, given symbols h of arity 3, f of arity 2 and constants a1 and a2 one can form the following terms : (f a1 a1), (f a2 a1), (h a1 a2 a1), (h a1 a2 a2) etc ... The set of flat terms is denoted by FT(F). The Working Memory is formalized as an ordered or unordered set of such terms.

The if-part of a rule (its left-hand side) is a conjunction of patterns, represented as a tuple (P1, ..., Pn). A pattern is a term some of whose arguments can be variable. Variables are denoted by X, Y, ..., they are taken from an enumerable set of variables V. Patterns are partial descriptions of facts. A pattern P matches a fact t if one can find a substitution of pattern's variables, σ : V → C, such that σP = t. For example the substitution of X by a1 and of Y by a2 in pattern (h X Y X) matches the term (h a1 a2 a1).

We say that the left-hand side of a rule (P1, ..., Pn) is instantiated (or that the rule is satisfied) when there exists a tuple of facts (t1, ..., tn) with ti ∈ WM, called the instance, and a substitution σ such that σPi = ti. We remark that since a pattern in a rule can match several facts in the Working Memory, a rule can be instantiated in multiple ways. The then-part of a rule (its right-hand side) is a sequence of actions that can add (resp. remove) a fact in (resp. from) the Working Memory.