ELITE アルゴリズム：確率コストモデルに基づくジョインネットの最適化方式

田野俊一[†] 増位庄一[†] 船橋誠壽[†]

推論速度の向上は，知識処理における最も共通かつ重要な課題として研究されている．現在，ルールベース型の知識処理システムにおける高速推論アルゴリズムとして，Rete アルゴリズム，TREAT アルゴリズムをはじめ，種々のアルゴリズムが提案されているが，特定の場面では，それぞれのアルゴリズムは効率的であるものの，逆に別の場面では，極端に効率を落としてしまうことが問題となってきている．本論文では，従来の高速推論アルゴリズムの問題点が，「パターンマッチ処理の履歴情報の記憶・利用レベルは，対象とする問題の性質により決めるべきであるのに対し，従来方式では，これが固定されている」ことにあることを示した．そこで推論処理を高速化するために，(i)実問題での推論処理過程をモニタし，そのログデータを解析し対象とする問題の性質を導き，(ii)すべてのジョイン構造とすべての履歴情報の記憶・利用レベルの組合せから最適なパターンマッチ処理構造を導くことにより高速な推論機構を実現する Elite (Eliminate Wasteful Memory Nodes by Optimizing Network Structure) アルゴリズムを提案した．問題の性質を表すパラメータをランダムに変化させた多くのシミュレーションの結果，Rete でもなく Treat でもない Elite 型のネットが最適なネットワークの多くを占め，また効率化の比率も大きいことが判明した．

ELITE: Eliminate Wasteful Memory Nodes by Optimizing Network Structure

Shun'ichi Tano,[†] Shoichi Masui[†] and Motohisa Funabashi[†]

As the importance of the rule-based system architecture is widely recognized, many researchers have studied a fast pattern matching algorithm to improve the inference speed. There are many algorithms, such as Rete algorithm, Treat algorithm and so on, however, they work effectively in quite restricted situations. Namely, since each algorithm was designed under specific assumptions, the algorithm cannot work well in the situations where the assumptions do not hold. In this paper, first, we show that the drawback of such algorithms comes from the fixed degree of usage of past pattern matching result. Second, we describe a fast pattern matching algorithm—Elite (which is short for "eliminate wasteful memory nodes by optimizing network structure"), for rule-based systems to resolve the drawback of conventional algorithms. Elite algorithm consists of two steps, (i) the characteristic is analyzed by monitoring the real problem solving process, (ii) the most effective calculation network is deduced from all join networks which have variable degree of usage of past pattern matching result by the characteristic given at the previous step. Finally, many simulations at various situations are done to show the effectiveness of our algorithm. The result proves that Elite algorithm is much more effective than conventional algorithms, such as Rete, Treat, in much wider situations.

1. はじめに

専門家の持つ知識を計算機に移植し，高度な知的処理を可能とする知識工学[2]が注目を浴びている．専門家の持つ知識を大局的に把握することは困難であり，試行錯誤的に知識を収集しなければならない場合が多い．そのため，知識の追加・修正・削除が容易であるルールによる知識表現が，多くの知識工学ツールで用いられている[15]．

しかし，その動作原理は，状況を認識し行動するという人間の動作モデル—認識・行動サイクル—を基本としており，必要とする計算量が大きく，推論速度が低いという問題点を有する[6]．

このため，処理速度（推論速度）の向上は，知識処理における共通かつ最も重要な課題として研究されている．現在，知識処理システムにおける高速推論アルゴリズムとして，多くのシステムで用いられその高速性が実証されている Rete アルゴリズム[3]，Treat ア

† (株)日立製作所システム開発研究所
Systems Development Laboratory, Hitachi, Ltd.

ルゴリズム[7]をはじめ，種々のアルゴリズム[5),8)~13)]が提案されている．

われわれも，高速推論アルゴリズムとして Rete アルゴリズムの改良方式[14]を開発し，さらに，ST-NET アルゴリズムと呼ぶ高速双方向推論アルゴリズム[16),17)]を提案してきた．

ところが，設計・計画型等の複雑かつ大規模な知識ベースに対しては，現在多くの知識工学ツールが用いている Rete アルゴリズムを適用すると，推論処理の効率が大幅に低下するケースが発生している．

本論文では，この問題点が，条件成立判定順序と過去の判定結果の記憶量の不適切さに起因していることを示し，Elite (Eliminate Wasteful Memory Nodes by Optimizing Network Structure) アルゴリズムと呼ぶ，確率コストモデルに基づくジョインネットの最適化方式について述べる．

以下，次章では，従来の研究を概観し問題点を明確にする．第3章では，本アルゴリズムを説明し，第4章で評価を行い，第5章で研究成果をまとめる．

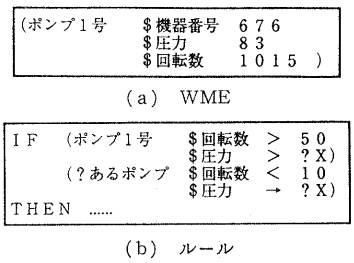
2. 従来の研究と問題点

2.1 従来の高速処理方式の分類[18]

まず，本論文で用いる典型的なプロダクションシステムの知識表現を簡単に説明する．

WME (Working Memory Element) は，**図1**(a)のように，フレームで表現する．この例では，フレーム名称が，“ポンプ1号”であり，項目として $ 機器番号，$ 圧力，$ 回転数を持ち，それぞれの現在値が 676, 83, 1015 である．

ルールはこの WME に関する記述より構成される．ここでは，1つの WME に関する条件記述を条件節と呼ぶ．“?”が付いた文字列は，変数であり，同一変数は同一値をとる．図1(b)のルール条件部は，「ポンプ1号の回転数が 50 より大きく，かつ，回転数が10より小さい“?あるポンプ”の圧力より，ポンプ1号の圧力が大きい」という条件をあらわしており，2つの条件節で構成されている．以下，条件節を，A，B，Cのように記述する．



(a) WME

(b) ルール

図1 知識記述の例
Fig. 1 Example of knowledge representation. (a) WME, (b) Rule

ルール条件を構成する個々の条件記述は，

(1) 1つの WME で条件成立判定ができる条件

(2) 複数の WME 間で条件成立判定ができる条件

に分類することができる．前者を，イントラ条件，後者をインター条件と呼ぶ．

図1(b)の個々の条件は，上記(1)(2)に対応して

(1) 1番目の条件節のイントラ条件

→ WME 名＝ポンプ1号

$ 回転数＞50

2番目の条件節のイントラ条件

→ $ 回転数＜10

(2) 1番目と2番目の条件節間のインター条件

→ 1番目の条件節の $ 圧力＞

2番目の条件節の $ 圧力

のように分類される．

パターンマッチとは，イントラ条件，インター条件を満たす WME の組を捜すことである．インター条件は，WME の組で条件判定を行う必要があるため，効率上「まず，イントラ条件を満たす WME を求め，次に，イントラ条件を満たした WME 間でインター条件の判定を行う」の順で処理する．

例えば，3つの条件節A，B，Cがあり，AとB，BとC，AとC間にインター条件がある場合，このパターンの満たすべき条件の構造は，例えば**図2**に示すようなツリーで表現することができ，これは，同時にパターンマッチの処理構造を示している．

すなわち，図2は，まずA，B，Cの条件節のイントラ条件を満たすかどうかをそれぞれ判定し，イントラ条件を満たした WME 間でインター条件の成立を判定すること，およびインター条件の成立判定順序が，「AとB間」，「AとC間，BとC間」の順である

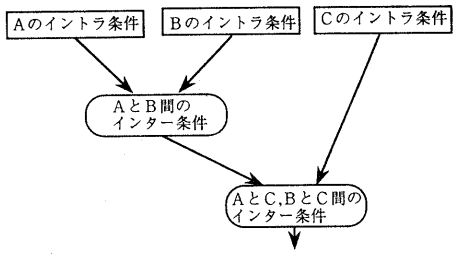


図2 条件判定の処理構造
Fig. 2 Calculation flow of matching.

ことを示している．

パターンマッチ処理の中間結果の記憶という観点で，従来方式を分類すると**図3**のようになる．方式0は，過去のパターンマッチ処理の途中結果，最終結果（コンフリクトセット）のいずれも記憶しない．方式1は最終結果のみを記憶し，過去のパターンマッチ処理の途中結果を記憶しない，方式2は最終結果とイントラ条件の判定結果を記憶し，インター条件の判定結果は記憶しない．方式3は最終結果，イントラ条件・インター条件の判定結果も記憶する．

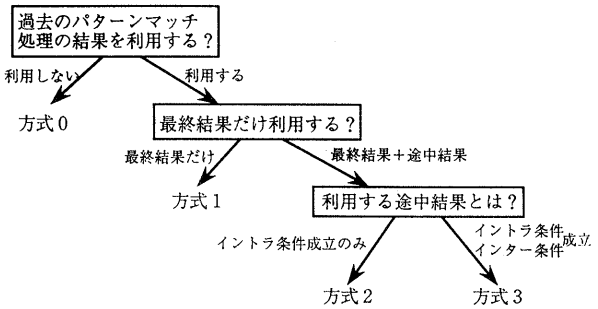


図3 従来方式の条件判定の処理履歴記憶／利用レベルによる分類

Fig. 3 Categorization of conventional matching algorithms.

方式3は最も有名なアルゴリズムであるReteアルゴリズム[3]であり，その改良アルゴリズムは多くのシステムで用いられている．方式2はReteより高速であると主張しているTreatアルゴリズム[7]である．

方式1と0は，以前は高速化をやっていないシステムに採用されていただけだったが，この方式の方が効率がよいとして，この方式を採用するシステムも現れている．例えば，KORE/IE[13]，XC[10]などである．

以上のように，従来方式は，過去の条件成立判定の処理履歴をどの程度記憶し利用するかにより方式0から方式3の4種に分類できる．

2.2 処理履歴の記憶レベルとジョイン順序の問題点

処理履歴の記憶レベルの問題

サーベイ論文[18]で示したように，従来の方式は処理履歴の記憶レベルにおいて問題を有する．

つまり，前節で示した各方式に対して，例えば，方式3，方式2，方式1，方式0の順で効率がよいとは一概にはいえない，またどの方式が最も高速であるかをいうこともできないのである．

例えば，方式1，2，3は，過去の処理履歴を記憶するが，記憶する手間より再計算の手間の方が少ないこともありうるし，また，対象世界の変化が激しく，過去の処理結果が次の時点では全く利用できない場合もありうる．

しかし，現在では，それぞれの方式を採用したシステムはそれぞれの得意なパターンマッチ処理となるアプリケーションを用いて性能評価を行い，自分が最も効率がよいと主張している．これは，ReteとTreatの比較（論争）[4),8),9)]が象徴的である．

Reteは，インター条件成立までの処理履歴を記憶するが，インター条件はWMEの組に対する条件であるので，履歴情報の記憶・管理に要する計算量は大きく，この情報がうまく利用されない場合，処理性が大きく低下する．

この問題点に対して，Treatは，インター条件成立の処理履歴を記憶しないことで対応している．この対応では，確かに，Reteで処理性が大きく低下したような部分に対しては，問題解決となっているが，一方で，Reteが効率よく処理していた問題に対しては，逆に処理性を大きく落とすことになる．

したがって，従来方式では，各方式で採用している履歴情報の記憶・利用レベルがたまたま問題と合えば効率よく処理できるものの，合致しないパターンに対しては逆に処理性を悪くする結果となる．

一般に，対象とする問題ごとに「適切な処理履歴の記憶・利用レベル」は異なると考えられるから，上記のような「方式2が最適である」，「方式3が最適である」といった議論は対象とする問題を決めなければ意味がなく，一般論としては議論できない．

そこで，文献18)では，問題を分析し適切な記憶レベルを見極めパターンマッチアルゴリズムを設計することが重要であると指摘した．

ジョイン順序の問題

また，最適な履歴情報の記憶・利用レベルは，条件判定の順序と独立に議論することはできない．条件判定順序は，図2で示したように，1種の木構造として表現できるのでここでは，「条件判定順序」をアークが合流する順序という意味で「ジョイン順序」と呼ぶことにする．

例として**図4**（a）に示すような3つの条件節からなるルールを考える．3つの条件節間にはすべてインター条件があり，このルール条件部の性質として

（1）条件節Aのイントラ条件をマッチするWME

はほとんど発生しない

(2)条件節B,Cのイントラ条件をマッチするWMEは頻繁に発生する

が成立するものとする.

この性質をもつ条件を図4(b)のようなネットワーク構造で条件成立判定を行う場合,性質(2)により条件節BとCの間のインター条件を満たすWMEの組は頻繁に生成されるが,性質(1)により条件節Aのイントラ条件を満たすWMEがほとんど発生しないので,頻繁に生成される条件節BとCの間のインター条件を満たすWMEの組は,そのとほんどが利用されない.したがって,図4(b)の矢印で示したところで条件節BとCの間のインター条件を満たすWMEの組を記憶することは意味がない.

つまり,図4(b)のネットワーク構造であればインター条件を成立したWMEを記憶することは避けなければならない.Rete型よりもTreat型の処理が好ましいことになる.

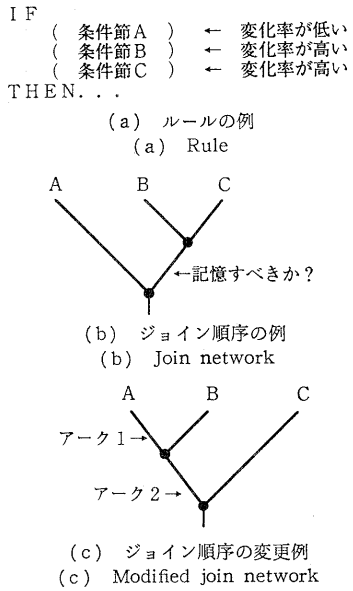
ところが,図4(c)のようにネットワーク構造を変更した場合,上記の例のように,無意味な条件節BとCの間のインター条件を満たすWMEの組の生成は行われず,条件節Aのイントラ条件を満たしたWMEが発生しない限り,インター条件のチェックが行われない.

図4(c)の構造では,ReteとTreatのいずれがよいかは,性質(1)(2)だけでは判定できず,さらに問題の性質を分析する必要があるのである.

すなわち,ネットワーク構造として図4(b)だけを考えた場合は,ReteよりTreatが効率がよいという結果になってしまうが,図4(c)のネットワーク構造も考慮に入れるとその結果は誤りであることがわかるのである.

以上のようにジョイン順序と履歴情報の記憶・利用レベルは関連が深く,両者を組合せて解析しなければ最適な条件成立判定の処理構造を求めることはできない.

これらの問題の部分的な解法がいくつか提案されている[1),5)].例えば,ジョイン順序を最適化する方式[5)]が提案されているが,Rete型のネットワークだけが対象になっている.つまり,記憶レベルは考慮されていない.また,記憶レベルをRete型とTreat型との切り替えができるようにした方式[1)]が提案されているが,このような単なる切り替えだけでは不十分であり,さらにジョイン問題も考慮されていない.



(a) ルールの例
(a) Rule

(b) ジョイン順序の例
(b) Join network

(c) ジョイン順序の変更例
(c) Modified join network

図4 ジョイン順序と履歴情報の記憶レベル
Fig. 4 Join sequence and storage of past result.

2.3 解決すべき課題

従来方式の問題点は,「履歴情報の記憶・利用レベルが固定化されている」ことにある.その結果,解くべき問題と各方式の履歴情報の記憶・利用レベルがミスマッチし,処理の効率を大幅に落としていた.

また,最適な履歴情報の記憶・利用レベルの問題は,ジョイン順序に大きく左右され,両者を組み合せて解析しなければ,最適な条件成立判定の処理構造を導き出すことはできない.

つまり,「ジョイン順序+履歴情報の記憶・利用レベル」の組が条件成立判定の処理構造を表すことになり,最適な「ジョイン順序+履歴情報の記憶・利用レベル」を導き,推論処理を高速化することが解決すべき課題である.以下,この「ジョイン順序+履歴情報の記憶・利用レベル」の組で表される条件成立判定の構造を「パターンマッチの処理構造」と呼ぶ.

3. Eliteアルゴリズム

3.1 アプローチ

従来方式の欠点は,前章でまとめたように履歴情報の記憶・利用レベルが固定化されていることであるが,「あるパターンは方式0,あるパターンは方式1…」のように単なる方式0,1,2,3の組合わせだけでは解決できない.つまり,従来方式で記憶する履歴情報は,(1)なし,(2)最終結果,(3)最終結果+イ

ントラ条件成立，(4)最終結果＋イントラ条件成立＋インター条件成立の4種に分類されるが，この組合せだけで十分とはいえない．例えば，(最終結析果＋インター条件成立) のような記憶の仕方も考えられる．

また，このような最適化を図るためには，問題の性質に関する情報が不可欠である．一般に，知識ベースシステムは，外界から解くべき問題や，データを与えられながら推論（問題解決）を行うので，知識として記述された WME やルールを静的に解析して問題の性質を導くことは困難である．

そのため，Elite アルゴリズムでは，ルールの静的解析から得られる情報（条件節の数や種類）に加え，処理系自身やハードの性質で定まる基本的な処理コスト，および，実際の推論過程をモニタすることにより得られる問題の統計的な性質（条件判定の成功確率など）から最適な処理構造を導く．

すなわち，

(1)実問題での推論処理過程をモニタし，そのログデータを解析し問題の性質を導き，

(2)すべてのジョイン順序とすべての履歴情報の記憶・利用レベルの組合せから最適なパターンマッチ処理構造を導く

ことにより高速な推論を実現する．

以下3.2節では処理構造の表現法について説明し，3.3 節で，1つの処理構造に対する処理コストの算出法について説明する．基本的には3.2節で得られるすべての処理構造についてコスト計算を行い，最適解を求めれば良いが，その組合せの数は莫大なものとなるため，その効率化について3.4節で説明する．

3.2 パターンマッチ処理構造の表現

Elite アルゴリズムを実現するためには，まず，パターンマッチの処理構造（ジョインの順序＋履歴情報の記憶レベル）のバリエーションを明らかにする必要がある．

基本表現

パターンマッチの処理構造を規定するのは

(1)ジョインの構造

「どの順序でインター条件を判定するか」

(2)履歴情報の記憶・利用レベル

「どの条件判定結果を履歴情報として記憶するか，しないか」

である．

(1)のジョイン構造は，木構造で表すことができる．木はノードの連結で表現できるのであるから，1つのノードに関する連結を表現できれば，それを再帰的に適用すれば木構造を表現できることになる．

つまり，あるノードが m 個の入力アークを持ち，入力アークの先に結合しているネットワークがそれぞれ N1, N2, …, Nm の記述で表現されたとすると，

(N1 N2 … Nm)

のリストによって，このノードを root とする木構造が表現できることになる．

したがって，木の root ノードから，この表現を，N1, N2, … に対して再帰的に繰り返せば，木構造全体が表現できることになる．リーフノードは，1つの条件節を表しており，この場合は，条件節の識別子で表現する．すなわち，条件節の識別子を要素とするリストとなる．

例えば，図4(b)，(c)の構造は，それぞれ，(A (B C))，((A B) C) で表現できる．

(2)の履歴情報の記憶レベルは，(1)のジョイン順序を表す木構造の各アークを流れる情報（WME の組）を記憶するかどうかで表現することができる．

例えば，上記の図4(c)の例では，アーク1を流れる情報を記憶することは，条件節 A のイントラ条件を満たした WME を記憶することに対応し（α-mem ノードに対応），アーク2を流れた情報を記憶するとは，条件節AとB間のインター条件を満たした WME の組を記憶することに対応している（β-mem ノードに対応）．

これは，上記で示したリストに「処理結果を記憶する／しない」を表現する以下の2種類の括弧を用いることにより表現できる．

'{', '}' この括弧でくくられた条件判定の結果は記憶しない．

'[', ']' この括弧でくくられた条件判定の結果は記憶する．

例えば，図4(b)で示したジョイン順序で，Rete 型のように，イントラ条件を満たした WME およびインター条件を満たした WME の組，インスタンシエーションを記憶する場合は，[[A] [[B] [C]]] のように記述する．A，B，C それぞれを囲んでいる '[' と ']' は各条件節のイントラ条件判定結果の記憶を指定している．その外側の，A と B を囲んでいる '[' と ']' は，条件節AとB間のインター条件判定結果の記憶を指定しており，最も外側の，A，B，Cを囲んでいる '[' と ']' は，すべての条件を満たした WME，つまり，インスタンシエーションの記憶を指

定していることになる．図4(c)の場合は，同様に，[[[A] [B]] [C]] のリストで表現することができる．

以上の例は，すべて '[' と ']' で囲み，条件判定結果を記憶することを指定したが，'[' と ']' の代わりに '{' と '}' で囲むことにより記憶しないことを指定することができる．

例えば，[[A] {[B] [C]}] のリストは，BとC間のインター条件の条件判定結果を記憶しないことを示している．これは，ダイナミックルーティングを行わない Treat 型の処理構造を示している．**図5** に基本表現の例とそのネットワークを示す．

上記の2種の括弧によるリスト表現を基本表現と呼ぶ．

拡張表現

基本表現では，あるノードにおいて3分岐以上の木構造となる場合がある．このような場合，基本構造ではどの順序で条件成立判定を行うかが表現されていないことになる．

例えば，トップレベルで3つの要素を持つ {[1] [2] [3]} の場合，1と2を比較して3を比較するのか，1と3を比較して2を比較するのかが規定されていない．同様に [[[1] [2]] [3] [4]] の場合，1と2の比較結果と3を比較して4と比較するのか，1と2の比較結果と4を比較して5と比較するのか，などが規定されていない．

この処理順序は，一意に決まらないからこそ，このような表現となっている．もし，一意に決められるのであれば，基本表現で表すことができる．

この部分は，Treat でいえば，ダイナミックジョインにあたる部分であり，この部分が表す重要な側面は，基本表現では，1つのネットワークを表していたのに対し，複数のネットワークを表すという性質である．つまり，データがどの枝から流れてくるかによって処理の順序が変わるのである．

例えば，[1 2 3] とあった場合，

1からデータが流れて来た場合：((1 2) 3)

2からデータが流れて来た場合：((2 3) 1)

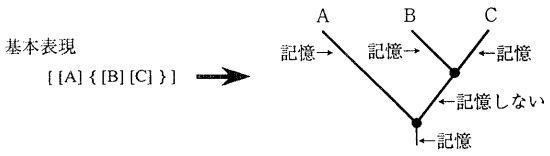


図5 基本表現の例
Fig. 5 Example of basic representation.

3からデータが流れて来た場合：((3 2) 1)

のように異なる処理形態をとる．

この並列構造の処理構造を以下のように，〈と〉で囲み表現する．これを拡張構造と呼ぶ．

[[1] [2] [3]]<((1 2) 3) ((2 3) 1) ((3 2) 1)>

ここで，〈 と 〉で囲まれた部分に書かれている数字は，条件節番号ではなく，並列構造内の順序番号である．例えば，

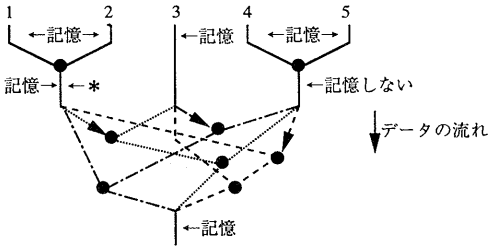
[[[1] [2]] [3] {[4] [5]}]

〈((1 2) 3) ((2 3) 1) ((3 1) 2)〉

の場合，拡張表現〈 〉内の番号1は基本表現内の [[1] [2]]，2は[3]，3は {[4] [5]} に対応している．**図6**(a)のネットワーク図では，拡張表現の3つのリスト ((1 2) 3)，((2 3) 1)，((3 1) 2) はそれぞれ点線，1点鎖線，破線で表される構造に対応する．それぞれのネットワークは，矢印に沿ってデータが流れて来た場合用いられる．例えば，新たな WME の組が，＊で示す枝を通過した場合は，点線で示した順序でパターンマッチが行われることになる．

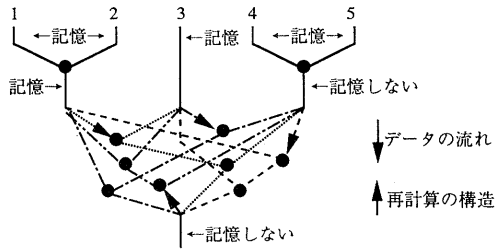
さらに，非記憶モードの場合は，再計算の処理順序も合わせて指定する．例えば，

[[[1] [2]] [3] {[4] [5]}] <((1 2) 3) ((2 3) 1) ((3 1) 2)>



(a) 拡張表現の例
(a) Extended representation

{[[1] [2]] [3] {[4] [5]}} <((1 2) 3) ((2 3) 1) ((3 1) 2) ((1 2) 3)>



(b) 再計算の情報を持つ拡張表現の例
(b) Extended representation with recomputation

図6 拡張表現とネットワーク構造
Fig. 6 Extended representation and the network.

{[[1] [2]] [3] {[4] [5]}

〈((1 2) 3) ((2 3) 1) ((3 1) 2) ((1 2) 3)〉

となり，拡張構造内の4番目の構造 ((1 2) 3) は，どのように再計算を行うかを示している．つまり，上記の条件節1，2，3，4，5からなるパターンの最終結果は記憶されていないので，このパターンの結果（この5つの条件節を満たす WME の組）を知りたい場合，再計算を行う必要があるが，その場合，((1 2) 3) の順序，つまり，図6（b）の2点鎖線で示すように [[1] [2]] の結果と [3] の結果とをまず比較し，次に {[4] [5]} の結果と比較する順序で再計算を行う．

この基本表現と拡張表現により，パターンマッチの処理構造を表現することができる．

3.3 パターンマッチ処理のコストモデル

処理構造を最適化するために用いる指標を**表1**にまとめる．これらの指標は，その性質により3種に分類できる．条件の個数：cond_n(i, j) は，ルールの条件部の記述から直接得られ，c，k，m はそれぞれイントラ条件，インター条件，記憶のための処理コストに関する指標でありインプリメンテーションやハードウエアに依存し，存在する WME の平均存在個数 w，成功確率 cond_p(i, j) は，実際の推論動作過程を解析することにより得られるものである．

本論文では，成功確率 cond_p(i, j) は条件判定の順序によらないと仮定している．

この指標を用いてパターンマッチの処理構造をどのように評価するかを例を用いて説明する．以下では，指標として表1のデータを用い，評価対象の処理構造として，[[[1] [2]] [3]] を取り上げる．

処理構造 [[[1] [2]] [3]] を通常のルールネットワークで表すと**図7**のようになる．ノード1，2，3はそれぞれ条件節1，2，3のイントラ条件を表している．表1のデータによれば comd_n(1, 1)=2 であるので，条件節1のイントラ条件は2つあるが，ここではノード1に2つのイントラ条件が入っていると考え，2つのノードに分けていない．

ノード4は条件節1〜2間のインター条件に対応し，ノード5は条件節1〜3間および条件節2〜3間のインター条件に対応している．具体的にはノード5は cond_n(1, 3)+cond_n(2, 3)=3 個の条件が入っている．

WME はルートノードから処理され，イントラノードでは条件が成立するものだけが次のノード（つまりインターノード）に流され，インターノードでは，条

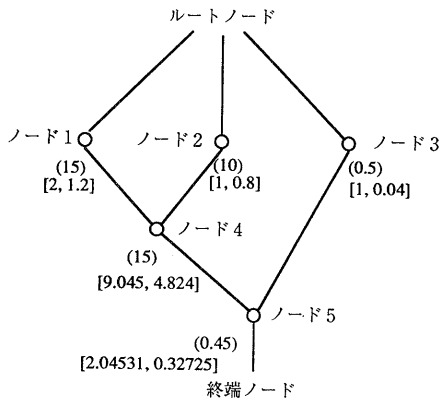


図7 コストの計算例
Fig. 7 Example of cost calculation.

表1 指標の意味とその例
Table 1 Meaning of indices and examples.

指　標	意　　味	例
cond_n(i, j)	$i \neq j$ の場合，条件節 i, j 間のインター条件の個数． $i=j$ の場合，条件節 i 内のイントラ条件の個数．	cond_n(1, 1)=2, cond_n(2, 2)=1, cond_n(3, 3)=1 cond_n(1, 2)=1, cond_n(1, 3)=2, cond_n(2, 3)=1
c	1つのイントラ条件判定の処理量	c=1
k	$\frac{\text{インター条件の処理量}}{\text{イントラ条件の処理量}}$ つまり，$k*c$ は1つのインター条件の処理量	k=3
m	$\frac{\text{長さ1のリストを記憶する処理量}}{\text{イントラ条件の処理量}}$ つまり，($m*c$) は要素数1のリストを生成し格納するための処理量．	m=8
w	知識ベースに存在する WME の平均個数	w=100
cond_p(i, j)	条件節 i と j 間の条件判定の成功確率． 条件節 i と j 間の条件が複数ある場合は複数個を同時に判定したときの確率．	cond_p(1, 1)=0.15, cond_p(2, 2)=0.1, cond_p(3, 3)=0.005 cond_p(1, 2)=0.1, cond_p(1, 3)=0.2, cond_p(2, 3)=0.3

件が成立する WME の組を作り，さらに次のノードへと流すという処理を行い，終端まで到達した WME の組が条件を満たす WME の組となる．

この処理構造は Rete 型であるからすべての中間結果は記憶される．つまり，ノード 1，2，3，4，5 の出力枝にはそこを流れた WME（の組）が記憶されている．

表 1 のデータにより，知識ベースに存在する WME の平均個数 $w=100$ であるので，100 個の WME がこのネットワークに流されて（登録されて）いることになる．例えば，ノード 1 の成功確率は，cond_p(1, 1)=0.15 であるので，ノード 1 の出力枝には 100*0.15=15 個の WME が記憶されていることになる．同様にノード 2 の出力枝には，100*0.1=10 個，ノード 3 の出力枝には，100*0.005=0.5 個が記憶されていることになる．ノード 4 では，ノード 1 と 3 から流れてきた WME 間の条件判定を行うが，その成功確率が cond_p(1, 2)=0.1 であるので，15*10*0.1=15 個の WME の組が記憶されていることになる．ノード 5 でも同様に，15*0.5*cond_p(1, 3)*cond_p(2, 3)=0.45 個記憶されていることになる．この値は図 7 に '(' と ')' で囲み示してある．

この状態において 1 つの WME をルートノードから流すために要するコストをこの処理構造のコストとする．

上記と同様に，ルートノードから 1 つ WME を流すと，ノードでは，1 つの WME が流れてきて，cond_n(1, 1)=2 個の条件判定を行わなければならない．表 1 のデータから 1 回のイントラ条件の判定には $c=1$ の計算量が必要であるので，比較のための計算コストは，1*2*1=2 となる．また，その結果，リストの長さが 1 でリストの個数が cond_p(1, 1)=0.15 のリスト列が生成されるので，記憶に要するコストは，0.15*1*(8*1)=1.2 となる．つまり，ノード 1 では，合計 2+1.2=3.2 の処理コストが必要となる．同様に，ノード 2，3 のコストも計算できる．

ノード 4 では，1 つの WME をルートノードから流すと左の入力枝から新たに 0.15 個のリスト，右の枝から 0.1 個のリストが流れてくることになる．したがって，比較のコストは，

$$(0.15*10+15*0.1+0.15*0.1)*3=9.045$$

となる．また新たに生成されるリストは，

$$(0.15*10+15*0.1+0.15*0.1)*0.1=0.3015$$

個であり，リストの長さは 2 であるので，記憶に要するコストは，2*0.3015*8=4.824 となる．同様に，ノード 5 のコストも計算できる．

図 7 には，各ノードごとに "[比較のコスト，記憶のコスト]" を示しておく．

これらをすべて加えると 22.2815 になり，これがこの処理構造を用いた場合のコストとなる．

次に，記憶しないモードを有する場合の処理の方法について説明する．[{[1][2]} [3]] のように，インターノード 4 の計算結果を記憶しない場合を例にとる．上記で計算したノード 4 での記憶のためのコスト 4.824 は不必要になる．したがって，図 7 に記載してある "[9.045, 4.824]" は "[9.045, 0]" となる．しかし，ノード 5 での処理ではノード 4 に記憶されているリストが必要となり，この場合，ノード 1 とノード 2 に記憶されている WME 間でインター条件処理（ノード 4）を行うことにより再構成する必要がある．1 回の再構成には，15*10*3=450 のコストが必要である．ノード 5 の右の枝からは平均 0.005 個のリストが流れてくるので，再構成に要する平均コストは，1 回の再構成に要するコスト*MIN(1.0, 0.005)=450*0.005=2.25 となる．

したがって，[{[1][2]} [3]] とした場合，ノード 4 のコストが 4.824 減少し再構成のためのコストが 2.25 増加するので，結局 2.574 の減少となる（つまり，[[[1][2]] [3]] より，[{[1][2]} [3]] の処理構造の方が効率がよい）．

残るは，拡張表現の部分，つまり，処理の順序が一意でない場合の処理であるが，これは単に指定された順序に従って上記と同様に処理をすればよい．例えば，図 6（b）の例では，条件節 3 から WME が流れてきた場合は，矢印に従って，1 点鎖線のネットワークに沿って，まず {[4][5]} の結果と比較し，次に [[1][2]] と比較するようにコストを計算すればよい．この拡張表現に対応するネットワークでは，中間結果は記憶されないので記憶コストは計算する必要はない．また，図 6（b）では再計算のための処理構造が拡張表現で与えられている．この場合も，単にこの構造，つまり，2 点鎖線のネットワークに沿って再計算を行えばよい．

以上のように，表 1 にまとめた指標を用いて，平均的な状態において 1 つの WME が変化したときに必要となる平均処理コストにより，処理構造を評価する．

例えば，**表 2** の指標が得られた場合，すべてのネッ

表 2 指標の例
Table 2 Example of indices.

指標	値
cond_n(i,j)	cond_n(i,j)=1（すべての i,j の組に1つ条件が存在）
c	c=1
k	k=2
m	m=200
w	w=1000

cond_p(i,j)

j \ i	1	2	3	4	5
1	0.020429	0.017676	0.152013	0.141105	0.161620
2		0.019507	0.114096	0.120866	0.018036
3			0.181890	0.175793	0.158049
4				0.133445	0.173565
5					0.030842

トワークを生成し，最良のネットワークを捜し出すと，**図 8** のようになる．この場合，最良のネットワークは RETE 型でも Treat 型でもないネットワークになり，そのコストは，最良の RETE 型のネットワーク，最良の Treat 型のネットワークの数分の1となっている．

3.4 最適処理構造の導出

まず，ジョインネットワークの総数を算出してみる．記憶モードを考えず，単純に2進木でジョインした場合，条件節の数が n のジョインネットワークの総数 $N(n)$ は，以下の漸化式で表現される．

$$N(n)=\sum_{i=0}^{n-2}\left\{\binom{n-1}{i}\times N(i+1)\times N(n-i-1)\right\}$$

$$N(1)=1$$

次に，N 進木とし各ノードの記憶モードも考慮に入ると，ジョインネットワークの総数は，以下の $B(n)$ となる．これは，本論文で示した基本表現の総数に対応する．

$$B(n)=\sum_{d=2}^{n}2\times B_d(n,d)$$

$$B(1)=2$$

$$B_d(n,d)=\sum_{i=0}^{n-d}\left\{\binom{n-1}{i}\times B(i+1)\times B_d(n-i-1,d-1)\right\}$$

$$B_d(n,1)=B(n)$$

さらに，本論文で示した拡張表現を含めると，条件節数 n のネットワークの総数 $E(n)$ は下式となる．これが，本論文が対象とするネットワークの総数となる．

$$E(n)=2\times E_d(n,2)+\sum_{d=3}^{n}(E_d(n,d)\times N(d)^d+E_d(n,d)\times N(d)^{d+1})$$

$$E(1)=2$$

$$E_d(n,d)=\sum_{i=0}^{n-d}\left\{\binom{n-1}{i}\times E(i+1)\times E_d(n-i-1,d-1)\right\}$$

$$E_d(n,1)=E(n)$$

表 3 に計算結果を示す．例えば，条件節数が5の場合，2種類の括弧で表される基本表現が 83,904 種類あり，この基本表現においてダイナミックジョインが必要な場合，'<' と '>' で囲まれた拡張表現を追加すると，4.329225e+13 通りの表現となることがわかる．N=105 の数は，Rete 型のネットワークの数に対応する．

最適な構造を算出する基本的な方法は，「ネットワークを生成し，それを評価する」であるが，表3の E の数字を見ればわかるように，なんらかの効率化を行わなければ，実際には算出不可能となってしまう．例えば，条件節数が5の場合，4.329225e+13 種類のネットワーク構造を生成し，それぞれについてコストを計算することになり，実行不可能である．

そこで，次章で報告するシミュレーションでは，以下の効率化手法を用いた．

表 3 ネットワーク構造の総数
Table 3 Number of network structures.

条件節数	N：2進木	B：基本表現のみ	E：拡張表現も含む
3	3	112	960
4	15	2592	12996480
5	105	83904	4.329225e+13
6	945	3490944	4.3118337e+22
7	10395	177497856	1.7452098e+34

最良のRETE ネット (コスト=97.9741)
[[[[[1] [2]] [5]] [4]] [3]]

最良のTreat ネット (コスト=169.254)
[[1] [2] [3] [4] [5]]
<((((1 2) 5) 4) 3) (((((1 2) 5) 4) 3) (((1 (2 3)) 5) 4)
(((1 (2 4)) 5) 3) (((1 (2 5)) 4) 3) >

最良のネット (コスト=33.3935)
[[[[1] [2] [5]]<((1 2) 3) ((1 2) 3) (1 (2 3)) > {4}] {3}]

図 8 表2の指標に対する最適なネットワーク
Fig. 8 The most effective network for the indices in Table 2.

表3をみればわかるように，B と E との差，つまり基本構造と拡張構造の個数の差が莫大である．これは，ダイナミックジョインのためのネットワークの組み合せの多さに起因している．例えば，[[1][2][3][4]] のネットワークには，4つのノードを起点とするダイナミックジョインがあり，ダイナミックジョインネットワークの総数（つまり，'<' と '>' で囲まれたネットワークの総数）は，上記の2進木の総数 $N(\cdot)$ を用いると，$N(4)^4$ になる．つまり，x 個のノードを起点とするダイナミックジョインの場合，拡張表現の種類は，$N(x)^x$ となる．

最適なダイナミックジョインのためのネットワークを生成するために，まず，[1] からデータが流れてきたときに用いるネットワークを生成し，次に，[2] からデータが流れてきたときに用いるのネットワークを生成し，,,のようにダイナミックジョインのための完全なネットワークの組み合せを生成した後評価するならば，上記のように，$N(x)^x$ 個のネットワークの組を生成しなければならない．しかし，例えば，[1]からデータが流れてきたときのネットワークは，[2]からデータが流れてきたときのネットワークに影響を与えない．つまり，各ジョインネットワークは独立に生成評価してもよい．この性質を利用すれば，場合の数を，$N(x)^x$ から $N(x)$ にすることができ，計算量が劇的に減少する．

また，表2において，N と B の差，つまり2進木と基本表現のネットワーク数の差も大きい．この差は，2進木から N 進木なったことよりも，2種の括弧を用いた記憶モードの設定の影響が大きい．そこで，記憶モードの設定の効率化も重要となる．ここでは，必ず記憶モードでなければならないノード，あるいは，必ず非記憶モードでなければならないノードを，そのノードの近隣のノードを流れる情報を基に判定するアルゴリズムを用いて，基本表現の場合の数を限定している．

具体的には，

(1) N 進木を生成

(2) 記憶モードを限定する．

　限定できないノードは場合分けする．

　ここで，基本表現が生成される．

(3) ジョインネットワークを独立に生成評価．

のように最適ネットワークを算出する．

4. シミュレーションによる評価

4.1 目的と方法

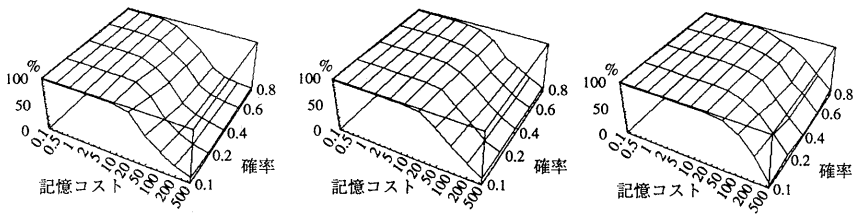
従来では，特定のアプリケーションを例にとり評価を行っていた．しかし，本論文の冒頭でも示したように，どの手法が有効であるかは，問題の性質に大きく依存するために，数種類のアプリケーションでの有効性を示してもあまり意味がない．

そこで，本論文では，表1に示した指標をランダムに変化させ，多くのシミュレーションを行い，これにより，Rete と Treat の比較，本方式の有効性検証を行った．

4.2 シミュレーション結果

Rete と Treat との比較

まず，従来から議論になっている Rete と Treat の比較を行う．条件節数＝4，c＝1，k＝2，cond_n(i, j)＝1 と固定し，記憶コスト m を 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500 の11通り，条件成立確率の上限を 0.1, 0.2, 0.4, 0.6, 0.8 の5通りとし，計 55 のパラメータの組を作成した．パラメータの各組において上記の条件成立確率の上限を満たすようにランダムに cond_p(i, j) を 100 ケース発生させシミュレーションを行った．各ケースにおいて Rete ネットワークと Treat ネットワークのどちらが効率的であるかを判定し，その割合を計測した．Rete が有効である割合を**図9**に示す．例えば，図9（a）において，一番手前の点（上限確率＝0.1，記憶コスト＝500 の点）の割合の値は 9% であるが，これは，条件節数＝4，w＝



（a） W=10　　（b） W=100　　（c） W=1000

図 9 Rete が Treat より有効である割合 (%)：条件節数 4
Fig. 9 Comparison of Rete network and Treat network.

10, $c=1$, $k=2$, cond_n$(i, j)=1$, $m=200$ の条件のもとで，cond_p(i, j) を 0.1 を越えないようにランダムに生成した 100 ケースに対しシミュレーションを行うことにより得られる．この場合，ランダムに生成した 100 ケースのうち 9 ケースでは最良の Rete ネットの方が最良の Treat ネットよりコストが小さかったことになる．

図 9 より，記憶コスト (m) が小さいときは Rete がよく，記憶コスト (m) が大きいとき Treat がよいということがわかる．またそのトレードオフ点は，ワーキングメモリの数 (w) によって，シフトしていることがわかる．

このことから，「Rete と Treat のいずれかが有効であるとはただちにいえず，m や w などのパラメータに依存する」ことがいえる．換言すれば，Rete と Treat はそれぞれ得意な領域があり，棲み分けが可能であることになる．ところが実際には，次に示すように Treat が有効な場面は非常に限られているのである．

Elite ネットの有効な領域

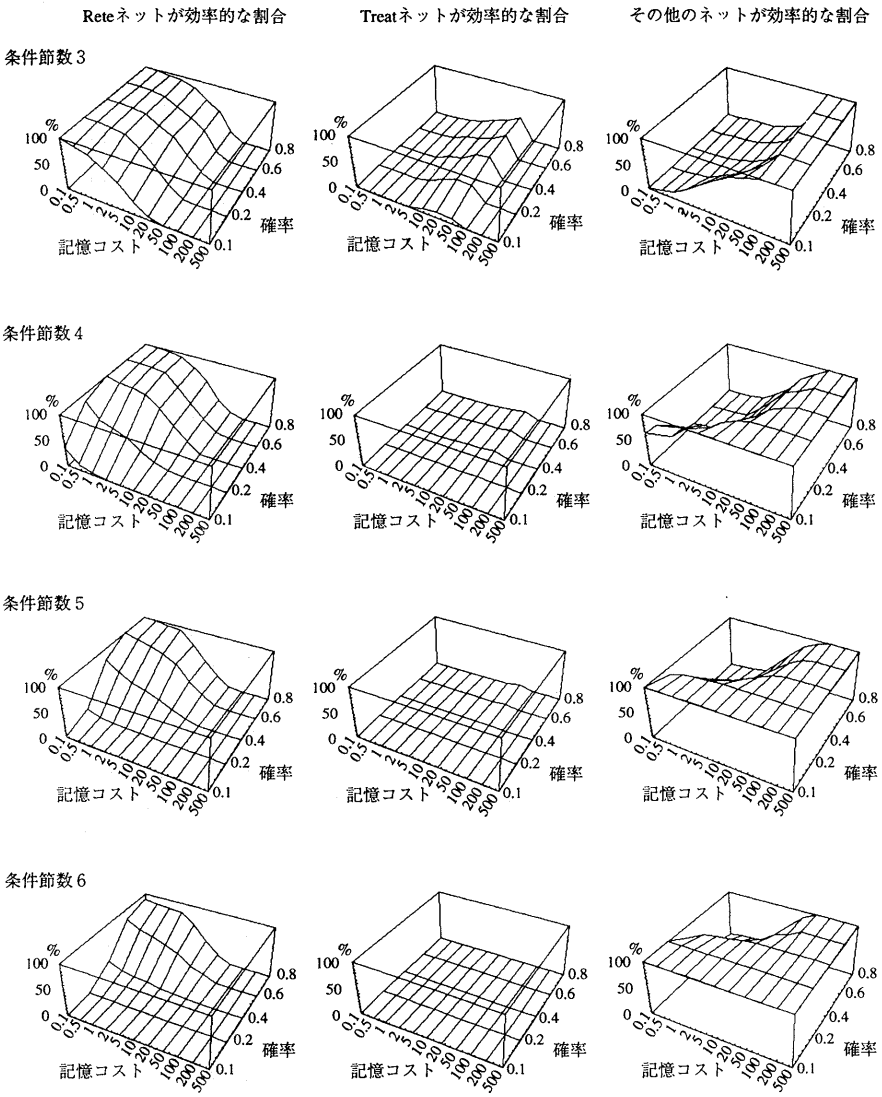
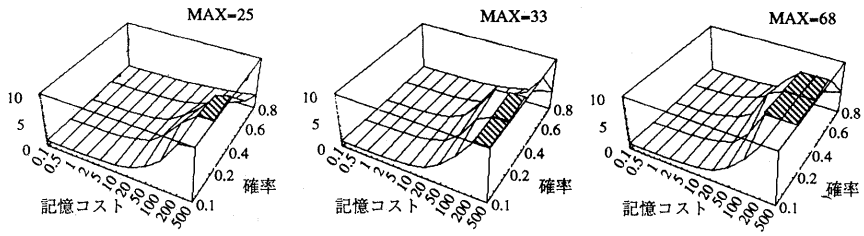


図 10 有効なネットワークの割合：$w=100$
Fig. 10 Ratio of the most effective network.



（a） 条件節数3
(a) 3 clauses
（b） 条件節数4
（b） 4 clauses
（c） 条件節数5
（c） 5 clauses

図 11 効率化比率：$w=100$
Fig. 11 Speedup factor.

次に，Elite ネットを考慮に入れてみる．Elite ネットは，Rete ネットや Treat ネットを含んでいるので，最適なネットは常に Elite ネットとなってしまうので，以下では，Rete ネット，Treat ネット，その他のネットのように3種に分類する．つまり，その他のネットとは，Elite ネットから Rete ネット，Treat ネットを除いたネットである．

図 10 に各ネットワークが効率的な割合を，$w=100$，条件節数が3から6までの場合について示す．グラフの軸の意味，実験方法は上記と同様である．

Rete ネットが有効である割合は条件節数が大きくなるにつれて小さくなっている．逆に，その他のネットは，条件数3でもかなりの割合を占め，条件節数が大きくなるにつれてその割合は大きくなっている．Treat ネットは条件節数が3の割合に多少あるものの，条件節数が大きくなるにつれて，その有効な割合はほぼ0になってしまう．

つまり，その他のネットの占める割合は，Rete と比較しても十分大きく，特に条件節数が大きい場合，その他のネットが最適なネットワークのほとんどを占めるようになる．

以上により，Rete でもなく Treat でもない Elite ネットが最適なネットワークの多くを占めることが判明した．これによって，Elite ネットの有効な領域が大きいことが判明した．次に，どれぐらいの処理コストの削減に貢献するのかを検討する．

Elite アルゴリズムの効率化の程度

図 11 に最適な Rete ネットワーク，最適な Treat ネットワークと最適な Elite ネットワークを求め，そのコストの比率を

$$\frac{\min\begin{pmatrix}\text{最適な Rete のコスト},\\ \text{最適な Treat のコスト}\end{pmatrix}}{\text{最適な Elite のコスト}}$$

で算出し，その最大値をプロットした結果を示す．例えば，図8のパラメータの場合，最適な Rete ネットのコストは 97.9741，最適な Treat ネットのコストは 169.254，最適な Elite ネットワークのコストは 33.3935 なので，コストの比率は

$$\min(97.9741, 169.254)/33.3935 = 2.93$$

と算出する．つまり，従来の（Rete と Treat）の最適なネットに比べてどれぐらい効率的かを表す指標である．

図11は，コスト比10倍を上限としてグラフ化してあり，ハッチングしてある平面は，この上限を越えたためにクリッピングされたことを示してある．最大の倍率は，図の上 MAX=25 のように示してある．

このように，Elite 型のネットワークを導入することによりかなりの効率化が図られる．その効果は，条件節数が大きくなる，つまり，条件記述が複雑になるにつれて大きくなる．

5. ま と め

（1）従来の高速推論アルゴリズムの問題点は，「パターンマッチ処理の履歴情報の記憶・利用レベルは，対象とする問題の性質によりルール条件部を構成する部分パターンごとに決めるべきであるのに対し，すべての従来方式では，履歴情報の記憶・利用レベルが固定化されている」ことにあることを示した．

（2）最適な履歴情報の記憶・利用レベルの問題は，ルール条件の判定順序，すなわち，ジョイン順序に大きく左右され，両者を組合せて解析しなければ，最適な条件成立判定の処理構造を導き出すことができないことを明らかにした．

（3）最適な「ジョイン順序＋履歴情報の記憶・利用レベル」を導き，推論処理を高速化することが解決すべき課題であることを示し，これを(i)実問題での推論処理過程をモニタし，そのログデータを解析し対象とする問題の性質を導き，(ii)すべてのジョイン構

造とすべての履歴情報の記憶・利用レベルの組合せから最適なパターンマッチ処理構造を導くことにより高速な推論機構を実現する Elite アルゴリズムを提案した．

（4）パターンマッチの処理構造は，「木を構成するアークに記憶モードを設定した N 入力の木構造」に対応する，2種の括弧を用いた基本表現とジョインネットを表す拡張表現により表現できることを示した．

さらに，パラメータをランダムに変化させた多くのシミュレーションを行うことにより，以下を示した．

（5）Rete と Treat のいずれかが有効であるとはただちにいえず，m や w などのパラメータに依存する．しかし，Elite ネットを考慮にいれると，Treat が有効な場面は非常に少ない．

（6）Rete でもなく Treat でもない Elite ネットが最適なネットワークの多くを占めることが判明した．これによって，Elite ネットが有効な領域が大きいことが判明した．

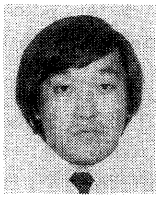
（7）Elite 型のネットワークを導入することによりかなりの効率化が図られ，その効果は，条件節数が大きくなる，つまり，条件記述が複雑になるにつれて大きくなる．

謝辞 本研究の場，およびその方向付けを与えてくださった，(株)日立製作所システム開発研究所 堂免信義所長，同明石吉三部長，ならびに有益なご意見をいただいた同関西システムラボラトリ古賀明彦研究員に深く感謝いたします．

参 考 文 献

1) 荒屋：状態保存式パターン照合アルゴリズムの定性的コスト分析と最良アルゴリズムの選択法，人工知能学会誌，Vol. 6, No. 3, pp. 435-439 (1991).
2) Feigenbaum, E. A.: The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering, *IJCAI-77*, pp. 1014-1029 (1977).
3) Forgy, C. L.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artif. Intell.*, Vol. 19, No. 1, pp. 17-37 (1982).
4) 稲葉，中村：マッチングアルゴリズム RETE と TREAT の比較，第37回情報処理学会全国大会論文集，pp. 1435-1436 (1988).
5) Ishida, T.: Optimizing Rules in Production System Programs, *AAAI-88*, pp. 699-704 (1988).
6) 増位，田野：プロダクションシステムの高速化，人工知能学会誌，Vol. 6, No. 1, pp. 38-46 (1991).
7) Miranker, D. P.: TREAT: A Better Match Algorithm for AI Production Systems, *AAAI-87*, pp. 42-47 (1987).
8) Miranker, D. P. and Lofaso, B. J.: The Organization and Performance of a TREAT-Based Production System Compiler, *IEEE Tr. on Knowledge and Data Engineering*, Vol. 3, No. 1, pp. 3-10 (1991).
9) Nayak, P., Gupta, A. and Rosenbloom, P.: Comparison of the Rete and Treat Production Matchers for SOAR (A Summary), *AAAI-88*, pp. 693-698 (1988).
10) Nuutila, E. et al.: XC—A Language for Embedded Rule Based Systems, *SIGPLAN NOTICES*, Vol. 22, No. 9, pp. 23-32 (1987).
11) Scales, D. J.: Efficient Matching Algorithms for the SOAR/OPS 5 Production System, Stanford Univ. KSL 86-47 (1986).
12) Schor, M. I. et al.: Advances in Rete Pattern Matching, *AAAI-86*, pp. 226-232 (1986).
13) 新谷：Prolog におけるプロダクション照合フィルタの高速化，情報処理学会論文誌，Vol. 32, No. 1, pp. 20-31 (1991).
14) 田野，増位，坂口，舩橋：知識ベースシステム構築用ツール EUREKA における高速処理方式，情報処理学会論文誌，Vol. 28, No. 12, pp. 1255-1268 (1987).
15) Tano, S., Masui, S., Nakano, T. and Mori, K.: EUREKA-II: A Programming Tool for Knowledge-Based Real Time Control System, *Proc. of Int. Workshop on AI for Industrial Applications 1988*, pp. 370-375 (1988).
16) 田野，増位：ST-NET アルゴリズム：双方向推論の高速処理方式，情報処理学会論文誌，Vol. 29, No 10, pp. 944-953 (1988).
17) 田野，増位，大森：高速双方向推論のための ST-NET 生成アルゴリズム，情報処理学会論文誌，Vol. 30, No. 9, pp. 1092-1102 (1989).
18) 田野：高速推論アルゴリズム，システム制御情報学会誌，システム/制御/情報，Vol. 33, No. 5, pp. 211-219 (1989).

(平成5年4月27日受付)
(平成6年1月13日採録)

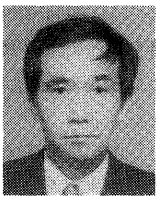


田野　俊一（正会員）

1958 年生．1981 年東京工業大学工学部制御工学科卒業．1983 年同大学院総合理工学研究科システム科学専攻修士課程修了．同年(株)日立製作所システム開発研究所入社．1990～1991 年カーネギメロン大学客員研究員．1991 年より国際ファジィ工学研究所勤務．人工知能，知識工学，自然言語理解，あいまい理論の研究に従事．人工知能学会，日本ファジィ学会，AAAI, IEEE 各会員．

増位　庄一（正会員）

1950 年生．1972 年京都大学工学部電子工学科卒業．1974 年同大学大学院修士課程修了．同年(株)日立製作所システム開発研究所入社．現在同所第 2 部主任研究員．人工知能，知識工学，エキスパートシステム，ソフトウェアエンジニアリングの研究に従事．著書「ビジネスマンのための AI 入門」(オーム社，共著)．AAAI, AIEEE 各会員．



舩橋　誠壽

1944 年生．1967 年京都大学工学部数理工学科卒業．1969 年同大学院修士課程修了（数理工学専攻）修了．同年(株)日立製作所入社．中央研究所配属．1973 年同社システム開発研究所の発足にともない転属，現在に至る．現在，同所主管研究員．1975～1976 年，MIT, スタンフォード大学客外研究員．1988～1992 年宇都宮大学非常勤講師，1992 年から東京工業大学非常勤講師．主に，システム制御技術，知的情報処理技術などの研究開発に従事．工学博士．計測自動制御学会，電気学会，IEEE, AAAI などの会員．