知識工学と人工知能 50－7
（１９８７．１．１７）

知識の構造化に関する考察

荒屋　真二
（福岡工業大学　通信工学科）

プロダクションシステムの推論高速化のための手法として、Ｒｅｔｅネットワークに代表されるように、知識の構造化があり、その有用性は高く評価されている。しかし、試行錯誤的開発スタイルをとる知識ベースシステムでは、前処理として行われる知識構造化の負荷が開発効率に大きな影響を及ぼすため、その負荷を減少させる必要がある。本論文では、知識の構造化処理自体の効率化を図るために、既存の構造化ネットワークを有効に利用し、それを部分的に修正する方法を提案する。この方法の基本は、ルールを構成する条件パタンや動作パタンをトークンとして構造化ネットワークに流してやりながら、ネットワークの修正に必要な情報を入手するという点にある。具体例として、Ｒｅｔｅネットワークにおけるルールの追加／削除時の処理方法について述べる。また、この考え方が、Ｒｅｔｅネットワークから連想ネットワークを効率的に生成することにも応用できることを示す。さらに、構造化レベルには知識ベース開発効率の点から最適値が存在し、それが開発の進展とともに動的に変化することを示す。

Consideration about Structuring Knowledge

Shinji ARAYA
Fukuoka Institute of Technology
3-30-1, Wajirohigashi, Higashi-ku, Fukuoka, 811-02, Japan

Structuring knowledge base, as in the Rete network, has been highly evaluated as a useful approach to efficient inference in production systems. However, knowledge base systems have a trial and error development style and the preprocessing load for structuring gives great influence on development efficiency of knowledge bases. Hence, it is necessary to reduce the preprocessing load. This paper proposes a method for making the knowledge structuring efficient which makes full use of, and partly modifies an existing structured network. The basic idea is that condition patterns and/or action patterns are flowed in the network as tokens in order to obtain the information necessary to modify the existing network. As an example, adding and removing of a production rule in the Rete network are presented. It is also illustrated that this idea is applicable to the efficient construction of the association network from the Rete network. Furthermore, it is shown that there exists an optimal structuring level from the view point of the development efficiency, and it changes dynamically together with development stage of knowledge base systems.

1。まえがき

プロダクションシステム（ＰＳと略す）の推論効率は、応答性が要求される実時間制御や対話型システムでの実用化の隘路になっており、知識ベースの大規模複雑化につれて益々その向上化が要望されている。そのため推論効率化を目指した研究が現在さかんに行われている。そのアプローチ法は、（１）知識ベースの構造化、（２）処理の並列化、（３）推論制御の高度化の三つに大別される。特に（１）の観点からの研究が多いが、これは（２）、（３）の基礎となるものであり、３者を併用することによって推論効率を一層向上させることができるからである。知識の構造化の大部分は推論実行に先だつ前処理として行われるが、その負荷の増大は知識ベースの開発効率に大きな影響を及ぼすようになる[4]。これは、段階的かつ試行錯誤的開発スタイルをとることの多い知識ベースシステムでは極めて重要な意味をもつ。しかし、構造化処理自体の効率化に関する研究は、筆者の知る限りではいまだ発表されていない。従来研究の大部分は知識をどのような形で構造化し、どのように推論に利用するかについてのみ焦点が当てられていた。また、（２）の並列処理の研究でも構造化処理の並列化は等閑視されているのが現状である。（３）ではヒューリスティックなメタルールが利用されているが、その構造化に関する議論はなされていない。

本稿ではＰＳの効率化について知識の構造化という観点から考察する。ここでいうＰＳの効率化とは単に推論実行の効率化だけでなく、それを実現するために前処理として行われる知識構造化の効率化、さらには知識獲得の効率化をも含んでいる。そして知識の構造化という考え方がこれら３種類の効率化にとって極めて重要な基礎となることを示したい。また、知識の構造化は、推論と同様にパタンマッチがその基本的情報処理となっており、知的システムに不可欠の要素であることを主張する。知識構造化のプロセスは人間の学習（知識獲得）との類似点が多数存在し、興味深くかつ重要な研究テーマと思われる。なお、本稿ではメタルールを含んだ知識ベースの構造化については考察しない。

まず、次節でＰＳの効率性について考察する。３節では知識の構造化をプロダクションメモリの構造化とワーキングメモリの構造化に分けて述べたあと、知識の静的構造と動的構造、および構造化ネットワークの線形化について触れる。４節では、知識構造化処理の効率化手法として、（１）分割構造化法と（２）逐次構造化法を提案する。また、Ｒｅｔｅネットワークから連想ネットワークを作り上げる方法を提案し、両者の関係を明かにする。５節では開発効率を最大にする構造化レベルについて考察する。６節では、本稿をまとめるとともに、今後の課題について触れる。なお、本稿はＯＰＳ５の文法［２］とＲｅｔｅアルゴリズム［１］についての知識があることを前提に書かれている。

2。ＰＳの効率性

ＰＳの効率を論じる場合、その表現力（シンタクス）が同じならば、次の二つが重要である。

（１）推論効率
（２）知識ベース開発効率

これまでの研究の中心は（１）にあった。知識ベースシステム完成時の効率にのみ関心が払われていたからだろう。推論効率を上げるために試みられてきた種々の改良法の基本にあるのが次節で述べる知識の構造化である。この構造化処理の大部分は推論実行に先立つ前処理として行われる。ゆえに、知識ベースが大規模化し、構造が複雑になってくると前処理の負荷が増大してくる。段階的かつ試行錯誤的プログラミングスタイルをとる知識ベースの開発では知識の追加、削除の頻度が高いので、構造化に要する前処理時間の増加は開発効率を大きく左右するものとなる。なぜならば知識ベースの開発は次の三つの作業が基本となるからである。

（１）知識の追加／削除
（２）知識構造化処理の実行
（３）推論の実行と評価

（１）に関しては知識獲得支援システム、高機能エディタ、あるいは知識ベース管理システムなどにおいて研究がなされているが、知識の構造化との絡みでは議論されていない。（３）は推論の効率化が知識ベースの開発効率の向上にも寄与することを意味する。しかし、その短縮化の度合いよりも（２）の増加の度合いが大きくなると問題だろう。後述するように推論時間と構造化処理時間との間にはトレードオフ関係が存在している。これら一連の作業は知識ベース開発の初期段階ほど高い頻度で行われ、完成に近づくにつれて（１）、（２）の占める割合が減少する。ここでは、ＰＳの効率を論じる場合、単

に推論効率だけでなく知識ベース開発効率からの視点が極めて重要であることを述べるにとどめ、それへの対応策については後述する。

3。知識の構造化

ＰＳにおける知識の構造化とは断片的知識（ルールや事実）の相互関係を明示的に表現することであり、次の二つに大別して考えることができる。

（１）プロダクションメモリ（ＰＭと略す）の構造化
（２）ワーキングメモリ（ＷＭと略す）の構造化

ここで、ＷＭを介してのルール間の相互関係は（１）に含め、ＷＭとＰＭの相互関係ならびにＷＭと競合集合メモリとの相互関係は（２）に含めて考えることにする。なお、本節以降ではＰＳの文法としてＯＰＳ５で採用されているもの［２］を前提として考察を進めることにする。

3。1　PMの構造化

ＰＭの構造化とはこれに格納される個々のルールの相互関係を明示的に表現することである。ルールは条件部と動作部からなり、条件部は一つ以上の条件パタンから、動作部は一つ以上の動作項からなる。動作項の中にはあるパタンをＷＭに追加、削除するものがあるが、このパタンをここでは動作パタンと呼ぶことにする。条件パタンと動作パタンは共に変数を含む。ルールの相互関係というものは、より細かく見ると次の四つに分けられる。

（１）条件パタンの相互関係
（２）動作パタンの相互関係
（３）条件パタンと動作パタンとの関係
（４）動作パタンと条件パタンとの関係

上記（１）の相互関係を用いて知識を構造化することにより推論の高速化を図ろうとした研究は多い（例えば［１］、［４］、［９］）。この場合ＰＭはネットワークの形で構造化されるが、これを弁別（識別）ネットと呼んでいる。このネットワークが実行可能なルールを弁別（識別）するのに利用されるからである。

（２）はある動作パタンと他の動作パタンの関係を抽出して構造化を図ることであるが、これによる効率化の向上はあまり期待できないため報告も今のところ見あたらない。ただし、追加ルールと既存ルールとの整合性判断などの機能をＰＳに持たせる場合には、この観点から構造化しておくとその処理を高速化できると思われる。

知識ベースの構築にあたって入力されるルールはいくつかの条件パタンとそれが満足した時にＷＭに追加、削除すべき動作パタンとの関係を明示したものであり、上記（３）に属する相互関係である。これは知識ベース作成者が意識して入力しなければならない唯一の構造情報である。ゆえに、これは前述の構造化ネットワークに組み入れられる基本的情報である。この種の情報を計算機が自ら作り出すことは特殊な場合を除いて現段階では不可能に近い。

（４）の関係を用いて構造化を図ろうとした試みもあり［３］、連想ネットワークなどと呼ばれている［６］。しかし、そこでは他の構造化との有機的繋がりを十分生かしておらず、連想ネットワークとは別に弁別ネットも使うという無駄をしている。また、連想ネットワークを形成するための構造化処理についても触れられていないし、現在最高速と評価されているＲｅｔｅアルゴリズム［１］との関連性も全く議論されていない。ただし、この視点からの構造化は推論に必要なパタンマッチのいくつかをポインタをたどる処理に置換できるので有望なアイデアと言える。

上記（１）、（２）、（４）の関係は与件ではないので、もし構造化を図ろうとするならば計算機に処理をさせなければならない。なぜならば、ユーザにこれらの構造化を強いることは、ＰＳがもつ本来の利点をなくしてしまうことになるからである。

3。2　WMの構造化

ＯＰＳ５においてクラス名をフレーム名、属性名／属性値をスロット名／スロット値とみなせば条件パタンや動作パタンは一種のフレームと考えられる。よってＷＭ要素（ＷＭＥと略す）もフレームであり、ＩＳ－Ａスロットや ＰＡＲＴ－ＯＦスロットなどを用いて構造化を図ることも可能である。しかし、ＰＳにおいてこのようなＷＭの構造化がいかなるメリットを生みだすかはまだ不明であり、逆に推論効率を低下させる要因になる恐れがありそうである[7]。

ここで取り上げるＷＭの構造化とはＷＭ内部に限定された構造化ではなく、ＷＭとＰＭとの相互関係を明示的に表現することである。ＰＭ内の個々のルールはＷＭを介して連鎖的に選択、実行される。Ｒ

ｅｔｅアルゴリズムでは条件パタン集合とWME集合との関連性を常に更新、保持している［１］。具体的には各条件パタンが現在どのWMEによって支持されているか（活性化されているか）をポインタによって関連づけている［５］。これによって、WMの変化部分（追加あるいは削除されたWME）だけに対するパタンマッチで済ませられるようになる。ここで注意を要することは、単に条件パタン単位だけではなく、一つのルール内の複数の条件パタンが同時に支持されている場合にはこの情報も別途ポインタによって保持されているという点である。これは複数の条件パタンに渡って多くの同一変数を含む場合のパタンマッチ（単一化）の効率化に大きく寄与している。すなわち、過去に行った単一化処理の結果が全て記憶されているので、同じ単一化を繰り返すという無駄が全くなくなるからである。

競合集合は条件部を構成する全ての条件パタンが同時に支持されたルールの集合であり、実際に支持しているWMEと関連づけて記憶される。変数や比較演算子を含む場合、一般には同一のルールを複数組のWMEが支持する。これらの情報を常に保持することもWMとＰMの構造化と考えることができ、これによって競合解消に必要なデータをポインタをたどることによって迅速に入手することが可能となる。

競合集合が大きい場合には、それ自体を構造化することが推論の効率化につながる［５］。競合解消戦略には種々のものが考えられるが、それらは何らかの基準に基づいて競合ルールに順序づけを行う。その結果を用いて競合集合を構造化しておけば、競合集合の変化時に最初から順序づけを行う必要はなくなる。

WMの構造化というものを別の観点から見ると興味深いものがある。原型ＰＳではＰMとWMが完全に分離独立していたのに対し、WMの構造化はWMＥを関連するルールの条件パタンと連結してしまう。これは短期記憶（WM）が長期記憶（ＰM）の中に分散化され、両者が融合した形で存在することを意味している。そしてこの繋がり具合によって個々の知識の活性度が表現されている。競合集合は活性度が最大になったルールの集まりと言える。

３．３　静的構造と動的構造

ルールの自動生成といった学習機能を持たないＰＳでは人間が知識ベースの修正を行わない限りＰMの構造は変化しない。このようにＰMは静的構造をもつため人間の長期記憶と対比されている。一方、WMの構造は遭遇した問題状況によって初期化され、また問題解決の過程において頻繁に変化するといったテンポラリな性格をもち短期記憶と対比される。ゆえにWMは動的構造をもち、推論実行中にその構造は変化する。ＰMもより大きな時定数で考えればルールの追加、削除により構造は変化するので動的構造をもつと考えることもできる。知識ベース開発の初期段階であればあるほど、また、問題が複雑になればなるほどＰMの構造変化の頻度は高くなる。後述する逐次構造化はＰMを動的構造をもつものとみなしている。照合成功確率の統計データにより構造の微調整を行うという考えも［８］ＰMのミクロな動的構造化といえる。このように知識の動的構造化には前処理段階で行われるものと推論実行中に行われるものの２種類が存在する。

３．４　構造化ネットワークの線形化

WMに追加あるいは削除された動作パタンのインスタンシエイション（WME）はトークンとして構造化ネットワークに流されパタンマッチ処理を受ける。この場合、ネットワークの有効リンクにそってポインタをたどるという操作が頻繁に必要となるが、これを減らそうというのがネットワークの線形化である[1],[5]。ある分岐ノードから次の分岐ノードまでの直線的に並んだノード群をメモリ上に順番に並べて格納することによってこれが達成される。トークンは途中で分岐ノードに出会うが、逐次マシンではそれにつながる複数のノードのうち一つだけにしか流れないので、他はバックトラックのためにスタックに積まれる。トークンはパタンマッチ成功の可能性が残されている限り分岐ノードへバックトラックする必要があるからである。並列マシンでは分岐ノードでトークンのコピーを作り並列に流せるのでスタックもバックトラックも不要であり処理が高速化される。

４．知識構造化の効率化

知識の構造化処理自体を効率化することの重要性については前に述べた。ここではそのための方法を提案するとともに、その特徴について考察する。

４．１　分割構造化法

これまでのＰＳではＰM内の全てのルールをまと

めて構造化処理しており、いわば一括構造化である。これに対し、サブタスクに関連したルール集合毎に構造化処理を行い、推論実行時にそれらをリンクするようにすれば、ルールの追加／削除による構造化のやり直しは一部分で済み効率的である。これは一般のプログラミング言語における分割コンパイルの考え方と同じと言える。

しかし、この分割構造化法にはいくつかの問題点がある。すなわち、この場合の構造化は人間によって分割されたルール集合の範囲内だけで行われ、グローバルな意味での構造化とは言えない。特に類似のサブタスクが多い場合や、サブタスク間でのデータのやり取りが途中で頻繁に起こるような場合には十分な構造化を行えない。これをリンク時に処理するとすればその負荷は増大し、一括構造化を最初からした方が効率的になる恐れもある。また問題が大規模複雑な場合にはサブタスク自体のルール集合も大規模になるので、やはり構造化処理そのものの効率化が必要になってくる。

４．２　逐次構造化法

４．２．１　基本的考え方

現在のＰＳではルールベースにわずか一つのルールを追加する時でも構造化を全てやり直している。既存のルールベースが大規模な時には、その無駄は相当なものとなる。そこで、まず、人間がルールのような断片的知識を他人や書物から獲得する場合のことを考えてみよう。視覚や聴覚を通して入って来た知識は既存知識と意識的あるいは無意識に照合され、既存知識との関連性が明かになると納得し、その関連性も含めた形で適当な場所に記憶されると思われる。包含される知識や全く同じ知識などが既存知識の中に存在する時には納得の度合いはより強いものになるだろう。既存知識の中に関連するものが全くない場合には人間は納得した気分にはなれず、とりあえず仮説として既存知識とは独立した形で記憶に入ると考えられる。入って来た知識が既存知識と矛盾する場合には自分なりにどちらが妥当なのかを考えたり質問したりする。

さて、人間の知識獲得過程をこのように考えると、一つのルールが知識ベースに加えられる度に全ルールを最初から構造化し直すという従来の方法はかなり不自然なものと言えるだろう。ゆえに、ここでは上述のような人間の知識獲得過程を素直に模擬した逐次構造化法を提案する。ただし、知識の整合性に関する問題は難しく、かつこれ単独でも大きな研究課題であるので、これは人間に任せるものとする。さて、逐次構造化法の基本となる考え方は次の通りである。

「追加／削除されるルールを、その構成要素である条件パタンと動作パタンに分解し、これらの個々のパタンをトークンとして既存の構造化ネットワークに順次流してやる。その過程でパタンマッチを行いながら既存知識との関係情報を入手するとともに構造化ネットワークの修正を行う。」

以下では構造化ネットワークとして最も高い評価を得ているＲｅｔｅネットワーク［１］を取り上げ、ルールを追加／削除する場合の処理について述べる。

４．２．２　ルール追加時の構造化処理

ルールの追加にあたっては、その条件パタンをトークンとしてＲｅｔｅネットワークに流してやる。トークンは、推論実行時に流されるトークン（ＷＭＥ）と大体同じような処理を受ける。すなわち、Ｒｅｔｅネットワークのルートノードからトークンが送出され、各ノードにおけるパタンマッチに成功した時にのみ後続ノードに引き渡される。ここで異なる点は、ＷＭＥの属性値は全て定数であるのに対し、条件パタンは変数、比較演算子、ＡＮＤ／ＯＲ条件などを含む場合があることである。そこで、１入力ノードではトークンの属性値を全てシンボルとみなしてパタンマッチを行うことにする。そうすれば、ノードの属性値が定数でトークン側が変数を含む場合にはパタンマッチに失敗する。定数でかつその文字列が等しい時にのみパタンマッチが成功する。パタンマッチに失敗した時には一つ手前のノード（成功した最後のノード）の深さ（ルートノードの深さを０とし、一つ進む毎に１づつ増加する）とそのノードの場所をトークンのタグとして記憶し、バックトラックする。Ｒｅｔｅネットワークではルール間で２入力ノードを共用しないので、２入力ノードに到達した時にはその時点で失敗とみなす。推論時には２入力ノードに到着したトークンをノードメモリに格納したが、構造化処理時には格納しない。バックトラック後に再び失敗した時には、過去にタグに記憶しておいたノードの深さと今回の深さを比較し、もし深ければタグを今回の深さと場所に変更する。この処理によりトークンが表わす条件パタンと最も類似した既存の条件パタンに対応するパスの途中のノードの場所をタグとしてもつことになる。

次にタグが表わすノードからまだパタンマッチに成功していない部分の条件を表わすノードを新たに

作り、順次リンクで繋いでゆく。ここでの処理はＲｅｔｅネットワークを作る場合と同じである。こうして、条件パタンを完全に表現したネットワークを作り上げる。

以上の処理を追加ルールの全ての条件パタンに対して行うことにより、ルール追加後のＲｅｔｅネットワークができあがる。

４．２．３　ルール削除時の構造化処理

ルールを削除する時に、その条件パタンをルートノードから流してやることは追加する時と同じである。しかし、今度はノードメモリ（２入力ノードの二つの入口についている）を使用するので、最初にノードメモリを全てクリアしてやる必要がある。トークンには変数、比較演算子、ＡＮＤ／ＯＲ条件が含まれるので、これを追加時と同様に単なるシンボルとみなしてパタンマッチを行う。トークンが分岐ノードを通過した時には、分岐ノードの直後のノードをタグとして記憶する。トークンは一般に複数回分岐ノードと通過するが、その都度タグは更新される。２入力ノードに到達したトークンは対応するノードメモリに格納される。２入力ノードでは条件パタン間にまたがる同一変数のチェックを行うが、変数はシンボルとみなされるので変数名が同じであれば成功し、合成トークンが送出される。このように、トークンは推論時と同じ処理を受けながら最終的には全てのトークンの合成トークンが一つの端末ノードに到達する。削除したいルールがもともとない場合には、合成トークンが端末ノードに到達することはないのですぐにわかる。念のために動作部が等しいかどうかをチェックしてもよい。この方法では、既存ルールの変数名と削除ルールの変数名が異なっていても同一ルールとみなされる。

端末ノードに到着した合成トークンを構成する個々のトークンのタグは、最後に通過した分岐ノードの直後のノードを表わしている。このタグが示すノードとそれに入るリンク、ならびにそれ以降のリンクとノードを削除すれば、ルール削除後のＲｅｔｅネットワークになる。

これまでの考察から、逐次構造化の基本にはパタンマッチがあり、推論実行時の処理とかなり類似した情報処理過程であることがわかる。

４．３　連想ネットワークの構成法

ここでは前述の逐次構造化法の考え方が連想ネットワークを作る場合にも応用できることを示す。まず始めにＲｅｔｅネットワークを作り、それを出発点として連想ネットワークを作っていく。

推論の過程において、あるルールが発火してその動作パタンがＷＭに追加あるいは削除された時、それはトークンとしてＲｅｔｅネットワークをルートノードから流れ始める。ゆえに、推論前にあらかじめ全ての動作パタンをＲｅｔｅネットワークに流し、個々の動作パタンがパタンマッチに成功する可能性のあるパスを調べておけば、推論時にトークンを流すべき範囲を限定することができ、推論が高速化される。この考え方に基づき、以下で述べる連想ネットワークではＰＭの動作部と条件部の関係をネットワーク構造として明示的に表現する。

推論実行時には動作パタンの変数にある特定の定数が束縛されているのでトークン中に変数はなかった。しかし、動作パタンを直接トークンとして流す場合には、それに含まれる変数の取り扱いかたが問題となる。これに対する簡単な方法は、条件パタンとトークン（動作パタン）のどちらかかに変数がある時には、その属性名／属性値のパタンマッチに成功すると仮定することである。つまり、両者の定数部分だけのパタンマッチを行い、これに失敗しなければパタン全体としても成功する可能性があるからである。これは安全策であり、これによってパタンマッチに成功する可能性のあるパスを見落としてしまう危険性は絶対にない。この考え方だけでもかなりの無駄なパタンマッチを回避できると思われる。

この考え方に基づく具体的な処理方法を次に述べる。今、動作パタンＰをＲｅｔｅネットワークのある一つのノードＮａから流してやればパタンマッチに成功する可能性のある条件パタンを全て包含する場合、ノードＮａを動作パタンＰの連想ノードと呼ぶことにする。ここで、動作パタンを連想ノードから流しても成功する条件パタンが必ずしも存在するとは限らないことに注意したい。この場合には、この動作パタンの追加／削除がそれ以降の推論に影響を及ぼさないことを意味する。動作パタンＰの連想ノードのうち深さが一番深いノードを特に最小連想ノードと呼ぶことにする。最小を付けた理由は連想される（パタンマッチを行わなければならない）条件パタンの数が最小だからである。この最小連想ノードは、動作パタンをトークンとみなし、次のようにして見つけることができる。

（１）トークンをＲｅｔｅネットワークのルートノードから流してやる。
（２）分岐ノードに出会った時には、それに繋がる複数個のノードが互いに排他関係にあるならば成功するノードへ分流する。もし成功するノードがない時には連想ノードがないことを意味するので処理を終了する。排他関係にない場合には、その分岐ノードを最小連想ノードとして処理を終了する。
（３）分岐していないノードでのパタンマッチに

成功した時にはトークンを後続ノードへ流す。失敗した時には連想ノードがないことを意味するので処理を終了する。

（４）分岐ノードか否かにかかわらず、対応するトークンの属性値が変数である場合には、そのノードを最小連想ノードとして終了する。

（５）トークンが２入力ノードに到達したならばそれを最小連想ノードとして処理を終了する。

以上の処理を全ての動作パタンに対して行い、求まった最小連想ノードの場所をそれぞれの動作パタンのタグとして付けておく。こうしておけば、推論時にある動作パタンがWMに追加／削除された時に、ポインタをたどるだけで一挙にその動作パタンの最小連想ノードからトークンを流すことが可能となる。この方法では最良の場合、最小連想ノードは２入力ノードとなるので、変数束縛のチェックだけで済むことになる。

ここでは、最小連想ノードを各動作パタンに対して一つだけ記憶する方法を述べたが、複数個を記憶するようにすればさらにパタンマッチの範囲を狭めることができる。

以上、Ｒｅｔｅネットワークの逐次構造化法ならびにＲｅｔｅネットワークから連想ネットワークを作る方法について述べた。これらの方法では、知識を構造化するのに既存の構造化ネットワークを利用するという点がポイントである。すなわち、知識の構造化を図ることは、単に推論の効率化だけでなく、構造化処理自体の効率化にも役立つという点に注目すべきである。同様の考え方は、他の構造化ネットワークの逐次構造化にも応用できると思われる。また、条件パタンや動作パタンをトークンとして既存ネットワークに流してやる際に、追加ルールの妥当性を判断するのに役立つ情報を同時に入手することも可能と思われる。これにより単なる文法チェックだけでなく意味的なチェックもある程度可能となり、わざわざ推論を実行しなくてもエラーを発見できる可能性を高められる。これは現在ボトルネックとなっている知識獲得を効率化することに寄与する。このような、追加知識に対する妥当性の判断も含めた逐次構造化は、非常に基本的な学習過程であると考えられる。人間が行う学習の大部分は、書物や他人から与えられた知識を既存知識を用いて自分なりに解釈、判断、構造化して記憶することにより、新らしい状況に対処できるようにすることだからある。逐次構造化法のより進んだ、これらへの応用については別の機会に報告したい。

５．最適構造化レベル

本節では、推論効率と構造化効率とがトレードオフ関係にあることを述べるとともに、知識ベースシステムのライフサイクルに応じて構造化レベルを動的に調整することの有用性を考察する。

５．１　構造化効率と推論効率のトレードオフ

知識ベースの構造化を図ることにより推論効率を向上させることができるため、構造化ネットワークは次第に大規模複雑化してきている。それに伴って推論実行の前処理として行われる構造化に要する処理時間は増大する。そのため前節で提案したような構造化処理自体の研究が重要になってくるわけである。しかし、構造化レベルを上げていけば、やはり構造化処理時間は増加し、推論実行時間は逆に減少するというトレードオフ関係が存在する。ここでいう構造化レベルとは、ルール相互の関係をどの程度の詳細度で構造化ネットワークとして明示的に表現するかということを意味している。Ｒｅｔｅネットワークより排他ネットワーク［４］の方が、さらにそれよりは前述の連想ネットワークの方が構造化レベルは高い。ルールの相互関係を多様な角度からきめ細かく表現すればするほど構造化レベルは高くなる。

構造化レベルを上げることが推論効率の向上に寄与することは確かだが、その程度は対象とする知識ベースの特性に依存する。個々の条件パタン間に共通性や排他性があまり存在しなければ、Ｒｅｔｅネットワークや排他ネットワークはその効果が減少する。また、動作パタンと条件パタンとの間に共通部分が少なければ、連想ネットワークとして構造化してもパタンマッチの範囲をあまり挟小化することはできない。さらに、このような知識ベース特性は知識ベースの開発段階によって変化する。なぜならば、開発が進むにつれて新らしいルールが追加され、ルール数が増大するからである。

５．２　最適構造化レベルとその変化

構造化レベルを１とした時の構造化処理時間をｆ（１）、推論実行時間をｇ（１）とすれば、１の増大に伴ってｆ（１）は増大し、逆にｇ（１）は減少する。ゆえに、ｆ（１）＋ｇ（１）を最小化する構造化レベルが存在するはずである。しかし、一回の構造化処理に対して、推論実行は一般に複数回行われる。また、数ルールをまとめて追加してから推論を実行して評価を行うこともある。この場合には、逐次構造化を行っても一回も推論実行を行わないこともありうる。ゆえに、開発効率を論じる場合には、次式で表わされるような平均処理時間Ｔを考える必

要がある。

$$T = f(l) + n * g(l)$$

ここで、nは構造化処理一回あたりの平均推論実行回数である。この場合もTを最小化する構造化レベルが存在する。しかし、この最適構造化レベルを厳密に求めることは困難である。なぜならば、関数f、gがどんな形のものか、nの値がどのくらいなのかを求めるのが大変だからである。また、以下に述べるように、f、g、nが開発が進むに伴って変化するので、最適構造化レベルも変化することになる。

f、g、nが知識ベースシステムのライフサイクルに応じてどのように変化するかを構造化レベルlを固定して考えてみる。一般に開発は段階的に推進され、その初期段階ほどより試行錯誤が多いのでルールの追加、削除の頻度は高く、完成に近づくにつれて次第にその頻度は小さくなると思われる。ゆえに、構造化処理が実行される頻度は開発時間の経過とともに減少する。ただし、知識ベースの規模（ルール数）は開発が進むにつれて増大するので、それに伴って一回あたりの構造化処理時間fは増大する。一方、推論実行の頻度は、開発の段階（実運用に供される前の段階）では知識ベースのデバッグのために行われるので、ルールの追加／削除が頻繁に行われる初期段階ほど高く、次第に減少すると考えられる。ただし、これは一回の構造化あたりの推論実行回数が初期段階ほど多いという意味ではない。一回あたりの推論実行時間gは、ルール数が増加し、より複雑な問題を解くことの多い開発後期の方が大きいと思われる。システムが実運用に入る頃にはエラーはほとんど発生しなくなり、構造化処理の頻度はかなり小さくなるので、一般にnの値は大きくなる。実運用時の推論実行の頻度は応用分野によって大きく異なる。

6。あとがき

本稿で述べたことをまとめると次のようになる。

（1）ＰＳの推論高速化のために知識の構造化は有効であるが、構造化処理自体の高速化も知識ベースの開発効率の点から重要である。

（2）ＰＳの構造化をＰＭの構造化とＷＭの構造化に分けて論じ、構造化にあたっての着眼点を整理した。

（3）構造化処理の高速化のための手法として、分割構造化法および逐次構造化法の二つを提案した。

（4）ルールをいくつかのパタンに分解し、それらをトークンとして構造化ネットワークに流してやりながら、構造化ネットワーク自体を修正するための情報を入手するという方法は、逐次構造化や連想ネットワークの作成に有効である。

（5）知識の構造化レベルには、開発効率を最大にする最適構造化レベルが存在し、それは開発の進捗とともに複雑に変化する。

今後の課題は、次のようである。

（1）逐次構造化法を実現し、その性能を実験的に評価する。

（2）Ｒｅｔｅネットワークと連想ネットワークの性能を構造化効率および推論効率の両面から理論的、実験的に比較評価する。

（3）構造化の過程で知識の整合性を判断するのにに必要な情報を入手する方法を明かにする。

（4）ＰＳにおける知識の構造化および推論実行の両面でのハッシュ法の有効な利用法を確立する。

参考文献

[1] Fogy,C.L.: Rete: A Fast Algorithm for Many Pattern/Many Object Pattern Match Problem, Artif. Intell., Vol.19, pp.17-37(1982)

[2] Fogy,C.L.: OPS5 User's Manual, Department of Compt. Sci., Carnegie-Mellon Univ. (1981)

[3] 鶴田ほか：連想・選別型推論のアナロジーによるプロダクションシステムの高速実行方式、情報処理学会論文誌、Vol.26,No.4 (1985.7)

[4] 荒屋ほか：知識の排他性を利用したパターン照合アルゴリズム、情報処理学会、知識工学と人工知能研究会資料、49-4 (1986.11)

[5]荒屋ほか：mini-OPS5 の概要、電気関係学会九州支部第39回連合大会 (1986.10)

[6]佐伯ほか：メタレベルのプログラミング機能を持つ高速推論システム、電子通信学会、人工知能研究会資料 AI86-12 (1986)

[7]渡辺ほか：ＣＬにおけるルール指向プログラミング情報処理学会、知識工学と人工知能研究会資料、46-3 (1986.5)

[8]田野ほか：推論高速化のための弁別ネットワークの動的変形法、情報処理学会第33回全大 (1986)

[9] 田野ほか：知識処理ソフトウエアＥＵＲＥＫＡにおけるルールネットワークの効率化方式、情報処理学会第32回全大 (1986)