人　工　知　能 95－5
（1994．7．21）

プロダクションシステムの高コストルール対処法に関する一考察

非会員　阿部武彦*　　非会員　南保英孝**
正員　木村春彦**　　正員　武部　幹***

*　石川職業能力開発短期大学校，穴水町
**　金沢大学工学部電気・情報工学科，金沢市
***　金沢工業大学情報工学科，野々市町

プロダクションシステムには，条件を満足したかどうかを調べるのに膨大な時間とメモリを必要とする高コストルールの対処問題がある．本稿では，イントラコンディションテストを通過したワーキングメモリエレメントを属性単位に分解して管理することにより，高コストルールに対処する方法を提案する．また，条件照合に失敗するかどうかをいち早く識別する方法を提案する．

A Consideration on a Method for Dealing with Expensive Productions of a Production System

Takehiko Abe, Hidetaka Nanbo,Nomembers,
Haruhiko Kimura,and Tsuyoshi Takebe,members

*　Polytechnic College Ishikawa, Anamizu-machi, 920 Japan
**　Faculty of Technology,Kanazawa University,Kanazawa-chi, 920 Japan
*** Department of Information and Computer Engineering,Kanazawa Institute of Technology,Nonoichi-machi, 921 Japan

In this papaer, we present a match algorithm for dealing with expensive productions in production systems, and demonstrate it's ability. Expensive productions are rules which would be required the extraordinary time and space to match (for each rule, compare the left-hand side against the current working memory). And we propose the improvement of a direct match algorithm for production systems, using one dimensional arrays stored an attribute value of a working mamory element.

1 はじめに

プロダクションシステムは人間の断片的な知識を表現するのに馴染みが良いため、エキスパートシステム構築用として最も多く使われている。しかし、条件照合に時間がかかるため、これまで効率の良い条件照合アルゴリズムが追求されてきた。中でも、条件照合の計算コストが大きい高コストルールの対処法が問題となっている。これまで、高コストルールへの対処法として文献(1)~(6)などの方法が提案されているが、さらに新規性や有効性のある方法が追求されているのが現状である。

本稿で提案する方法は、属性値が重複しやすいときに効果を発揮するものであり、直接条件照合アルゴリズムの範疇に入る。直接条件照合アルゴリズムとは文献(6)で提案された高コストルール対処法であり、αメモリから条件要素を満足させるワーキングメモリエレメント(WME)を探し出して、そのWMEの属性値を条件要素の対応する変数に代入していくことにより、照合時に調べなければならないWMEの組合せの数を減らしていく方法である。αメモリとは、対応する条件要素の初期条件を満足させるWMEを記憶するメモリである。残念なことに、この方法ではまだ調べなければならないWMEの組合せの数が多く、照合の計算コストは大きい。また、計算コストの小さいルールに適用すると従来のReteアルゴリズム(7)よりも極めて効率が悪くなることが知られている。本稿では、直接条件照合アルゴリズムを改善する方法として、WMEを属性単位に分解して管理する対処法と、条件照合に失敗するかどうかをいち早く識別する方法を提案する。

「条件照合に失敗するかどうかをいち早く識別する」ことがなぜ効率向上につながるかの理由は、照合時に調べなければならないWMEの組合せの数を更に減らすことができるからである。また、このアイデアが今まで生まれなかった主な理由は、これまで主に使われてきたReteアルゴリズムが、WMEを分解することなく、ジョイン演算によりルールの条件を満足させるWMEの組を求めてきたからである。ここで、ジョイン演算とは、条件要素間で変数の値が矛盾しないかどうかを調べるテストである。

2 対象とするプロダクションシステム

本稿で対象とするプロダクションシステムはOPS5[8]である。OPS5は、現在最も広く使われている前向き推論方式のプロダクションシステム記述言語であり、完成度が最も高い。前向き推論とは、ルールの条件部とワーキングメモリ(WME)の間で照合を行ない、行動部で指示される内容に従ってWMの状態を書き換える推論方式をいう。WMEはクラス名と一連の属性で表わされ、属性は属性名と属性値の対で表わされる。属性値には定数が入る。また、WMEにはWMに追加されるときにタイムタグと呼ばれる整数が付けられる。この整数はだんだんと大きなものとなる。ルールは、

<ルール>:=(P <ルール名><条件部>→<行動部>)

で表現され、条件部は条件要素の連言である。条件要素はWMEと対応するパターン記述である。

本稿では、文献(6)と同じ制約を設ける。以下にその制約を示す。

(1) 否定型の条件要素は許さない。つまり、適合するWMEが存在しないという条件要素は許さない。

(2) 属性値には変数と定数だけを許す。つまり、条件照合は定数のマッチングのみである。

3 提案方法

WMEを属性単位に分解して管理する対処法を提案1とし、条件照合に失敗するかどうかをいち早く識別する方法を提案2として以下に示す。

3.1 提案1

高コストルールになりやすい場合として、属性値の取り得る値の数が少ない場合が上げられる。なぜなら

ば、このような場合には照合の成功率が高くなるので計算コストが大きくなるからである。いま、ある条件要素

(C1 ^a_{11} <X> ^a_{12} <Y> ^a_{13} <Z>)

を考える。ここで、C1はクラス名、a_{11}、a_{12}、a_{13}は属性名、X、Y、Zは属性値に置かれた変数である。この条件要素の各属性値の取り得る値の数を5とすると、この条件要素を満足させるWMEは最大125通りとなる。いま、この125個の異なるWMEが記憶されていると仮定する。また、同じルール内の他の条件要素でX、Y、Zの内の2つを含むものがあり、既に条件照合が終っていてこの2つの変数(仮にY、Z)の値が確定しているとする。このとき、残りの変数(X)の値だけが未定なので、上記の条件要素を満足させるWMEの候補は5個となる。あとは、候補となったWMEが存在するかどうかをハッシュ法を用いて調べればよい。これに対し、これまでの直接条件照合アルゴリズムだと、125個のWMEすべてに対して条件要素の条件を満足するかどうかを調べなければならない。むろん、条件照合が進むうちに変数の値が確定してくるので、条件要素の条件が制約されてくる。この例のように前者の候補数が後者の候補数と比べて圧倒的に少ない場合には前者の処理の方が計算コストが小さくなる。

本稿では、イントラコンディションテストを通過したWMEを属性値単位に分解して記憶する。ただし、対象となる属性値は条件要素の変数に対応する値だけであり、各変数ごとに同じ値が重複して記憶されないようにする。ここで、イントラコンディションテストとは、変数の値が確定していない初期の条件要素を、追加されたWMEが満足させるかどうかを調べることである。このテストに成功したWMEはαメモリに記憶される。　このようにすれば、値の確定していない変数についてだけ属性値を呼んできて候補のWMEを生成し、それが存在するかどうかをハッシュ法により調べることにより条件要素の条件を満足させるWMEを導くことができる。

提案1の条件照合アルゴリズムを図1に示す。以下にその概略を図中の番号順に従って説明する。

(1) ルールの実行により WM に追加/削除されるWMEwme と追加/削除を区別するデータ Z を入力する。

(2) 配列 TB を用いて、wme のクラス No. から、同じクラス No. の条件要素をもつルールとその条件要素が何番目のものかを表わすデータ (i,j) をすべて引き出す。ここで、i はルール No.、jは条件要素の序数詞を表わす。

(3) wme が削除された WME であれば (4) へ、追加されたWMEであれば (7) へ飛ぶ。

(4) WME を WM とαメモリから削除する。そして、CS メモリから wme を含むインスタンシエーションを削除する。ここで、インスタンシエーションとは、条件を満足したルールとその条件を満足させた WME の組の対である。

(5) wme と同じクラス名をもつ条件要素を含む全ルールに対して、(6) の削除処理を繰り返す。

(6) 配列TBによって引き出された条件要素 E* を wme が満足させるかどうか調べる。満足しなければラベル L0 へ飛ぶ。満足すればE*の変数に対応する配列 A_{ijx}、AP_{ijx}、スタック S_{ijx}から、変数に対応する wme 内の属性値を削除する処理を行なう。

(7) wme を WM に追加してから、wme と同じクラス名をもつ条件要素を含む全ルールに対して、以下の追加処理を繰り返す。

(8) 引き出された条件要素 E*に対応するαメモリに wme を追加する。

(9) E*の各変数に、wme の対応する属性値を代入する。ここでは、変数 x に代入される属性値を a_x で表わす。また、ルール内の同一変数にも属性値をコピーする。次に、E*の変数に対応する配列 A_{ijx}、AP_{ijx}、スタック S_{ijx}に対して、wme の対応する属性値を追加する処理を行なう。そして、

```
begin
(1) input wme,z;
   t ← class-No. of wme;
(2) detect a set of rules H with the same class-No. by t from TB;
(3) if z="deletion" then
(4)    begin
       delete wme in WM;
       delete wme in α-memory;
       delete instantiations including wme in CS-memory;
(5)     for R∈H do
        begin
          intracondition test of a condition element E* with a class-No. t;
(6)        if wme doesn't satisfy E* then go to L0;
          i ← rule-No. of R;
          j ← an ordinal number of E* in R;
          V← variable set in E*;
          for x∈V do
            begin
              A_ijx(a_x) ← A_ijx(a_x) − 1;
              {attribute-value-No. a_x in wme assigns to variable x}
              if A_ijx(a_x) = 0 then
                begin
                  S_ijx(AP_ijx(a_x)) ← S_ijx(P_ijx);
                  AP_ijx(S_ijx(P_ijx)) ← AP_ijx(a_x);
                  P_ijx ← P_ijx − 1
                end
            end
    L0: end
    end
  else z="addition"
(7)   begin
       add wme in WM;
       for R∈H do
         begin
           intracondition test of a condition element E* with a class-No. t;
           if wme doesn't satisfy E* then go to L1;
           i ← rule-No. of R;
           j ← an ordinal number of E* in R;
(8)        add wme in α-memory for E*;
(9)        copy each attribute-value-No. of wme to the corresponding
             variable of E*;
           {This operation assigns to variable x attribute-value-No. a_x}
           V← variable set in E*;
           for x∈V do
           if A_ijx(a_x) = 0 then
             begin
               A_ijx(a_x) ← A_ijx(a_x) + 1;
               P_ijx ← P_ijx + 1;
               AP_ijx(a_x) ← P_ijx;
               S_ijx(P_ijx) ← a_x;
             end
           else
             A_ijx(a_x) ← A_ijx(a_x) + 1;
(10)       NEW-COMBINATORICS(R,j,wme)
    L1: end
  end
end
```

図1　提案1のアルゴリズム

手続き NEW-COMBINATORICS(R,j,wme) を実行する。

(10) 手続き NEW-COMBINATORICS(R,j,wme) は、追加された WMEwme がルール R の j 番目の条件要素を満足させたときの、ルール R に対するインスタンシエーション
(R:$W_1, W_2, \cdots, W_{j-1}$,wme,$W_{j+1}, \cdots, W_e$) を求める手続きである。

(例 1) 次のルールを考える。

(P i
(C1 ^a_{11} <X> ^a_{12} <Y> ^a_{13} <Z>)
(C2 ^a_{21} <Z> ^a_{22} 10)
(C3 ^a_{31} <W> ^a_{32} <X> ^a_{33} <Y>)
→ actions)

ここで、i はルール No. 、C1,C2,C3 はクラス名、^a_{11},^a_{12},$\cdots$,^a_{33}は属性名、X,Y,Z,W は変数名である。

いま、次のような WME が追加されたとする。

10:(C2 ^a_{21} 3 ^a_{22} 10)

ここで、左端の整数 10 はタイムタグである。このとき、ルール i の 1 番目の条件要素は次のようになる。

(C1 ^a_{11} <X> ^a_{12} <Y> ^a_{13} 3)

この条件要素に対応するαメモリに次のような WME が入っていたとする。

1:(C1 ^a_{11} 2 ^a_{12} 3 ^a_{13} 3)
2:(C1 ^a_{11} 2 ^a_{12} 2 ^a_{13} 3)
3:(C1 ^a_{11} 2 ^a_{12} 2 ^a_{13} 2)
4:(C1 ^a_{11} 3 ^a_{12} 2 ^a_{13} 1)
5:(C1 ^a_{11} 3 ^a_{12} 3 ^a_{13} 2)
6:(C1 ^a_{11} 2 ^a_{12} 3 ^a_{13} 3)

このとき、配列 $A_{i1x}, AP_{i1x}, A_{i1y}, AP_{i1y}$とスタック S_{i1x}, S_{i1y}は図 2 のようになる。変数 X,Y の取り得る値の組合せから、キーは (3,2,3),(3,3,3),(2,2,3),(2,3,3) となり、この内、該当する WME が存在するのはタイムタグ 2 と 1 の WME だけである。3 番目の条件要素に対応するαメモリに次のような WME

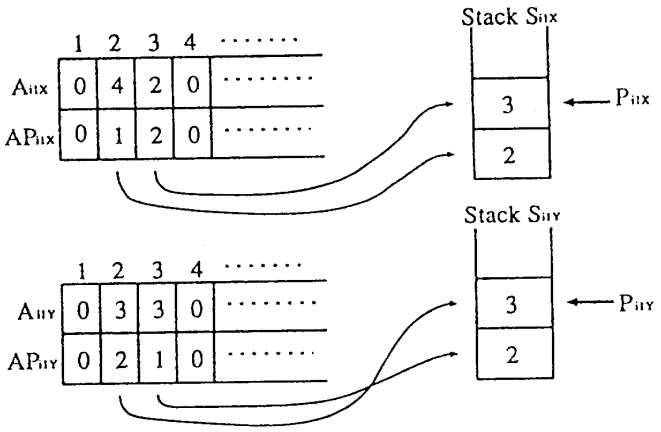

図 2　配列とスタックの内容

7:(C3 ^a_{31} 1 ^a_{32} 2 ^a_{33} 3)
8:(C3 ^a_{31} 2 ^a_{32} 4 ^a_{33} 2)
9:(C3 ^a_{31} 3 ^a_{32} 3 ^a_{33} 4)

が入っていたとすると、インスタンシエーションは (i:1,10,7) と (i:6,10,7) となる。

3.2　提案 2

高コストルールの条件要素内の各変数ごとに、属性値の度数を入れる一次元配列 A_{ijx}を用意する。ここで、
i:ルール No. 、j:条件要素の序数詞、x:変数名である。尚、この配列の添え字は変数の値を番数化したものである。

以下の操作で条件照合に失敗するかどうかをいち早く識別することができる。

(1) 追加された WME が条件要素を満足させるとき、条件要素の変数 Xに WME の対応する属性値 a_x を代入する。このとき、a_xを番数化し、それを添え字にもつ配列要素 $A_{ijx}(K)$ の値を 1 増やす。

(2) 同一ルール内で、Xをもつすべての条件要素に対して、条件を満足させる WME の中に、Xに対応する属性値が a_xである WME が存在するかどうかを配列 A_{ijx}を用いて調べる。もし存在するならば、a_xをすべての Xにコピーする。また、存在しないならば条件照合に失敗したことがわかる。

このチェックを照合前にしておけば、条件を満足するWMEが存在しないにもかかわらず、条件照合をさせる無駄を一部防ぐことが可能となる。

図3に提案2のチェックを組込んだ条件照合アルゴリズムを示す。文献(6)で提案された直接条件照合アルゴリズムと異なる点は、配列A_{ijx}を導入することにより、照合に失敗するケースをいち早く知ることができるようにしたことである。以下にその概略を図中の番号順に従って説明する。まず、

(1) ルールの実行により WM に追加/削除されるWMEwme と追加/削除を区別するデータ Z を入力する。

(2) 配列 TB を用いて、wme のクラス No. から、同じクラス No. の条件要素をもつルールとその条件要素が何番目のものかを表わすデータ (i,j) をすべて引き出す。ここで、i はルール No.、jは条件要素の序数詞を表わす。

(3) wme が削除された WME であれば (4) へ、追加された WME であれば (7) へ飛ぶ。

(4) wme を WM とαメモリから削除する。そして、CS メモリから wme を含むインスタンシエーションを削除する。ここで、インスタンシエーションとは、条件を満足したルールとその条件を満足させた WME の組の対である。

(5) wme と同じクラス名をもつ条件要素を含む全ルールに対して、(6) の削除処理を繰り返す。

(6) 配列TB によって引き出された条件要素 E* を wme が満足させるかどうか調べる (イントラコンディションテスト)。満足しなければラベル L0 へ飛ぶ。満足すれば E* の変数に対応する配列 A_{ijx} に対して、wme の対応する属性値の度数を 1 減らす。

(7) wme を WM に追加してから、wme と同じクラス名をもつ条件要素を含む全ルールに対して、以下の追加処理を繰り返す。

(8) 引き出された条件要素 E* に対応するαメモリへ wme を追加する。

(9) E* の各変数に、wme の対応する属性値を代入する。ここでは、変数 x に代入される属性値を a_x で表わす。次に、E* の変数に対応する配列 A_{ijx} に対して、wme の対応する属性値の度数を 1 増やす。そして、手続き COMBINATORICS*(R,j,wme) を実行する。

(10) 文献 (6) の手続き COMBINATORICS(R,j,wme) に PRECHECK*(t,L) を加えたものである。PRECHECK*(t,L) は t 番目の条件要素を満足させた WME の属性値 a_x をその条件要素の「値が未確定の変数 x」に代入する操作の後に行われる。まず、同一ルール内で、x をもつすべての条件要素に対して、条件を満足させる WME の中に、x に対応する属性値が a_x である WME が存在するかどうかを配列 A_{itx} を用いて調べる。もし存在するならば、a_x をすべての x にコピーする。また、存在しないならばラベル L へ飛ぶ。

(例 2) 次のルールを用いて PRECHECK*(t,L) の操作例を示す。

```
(P r
   (C1 ^a11 <A> ^a12 <B> ^a13 <C>)
   (C2 ^a21 <D> ^a22 <C> ^a23 T1)
   (C3 ^a31 <E> ^a32 <D> ^a33 <F>)
   (C4 ^a41 T2 ^a42 <E> ^a43 T3)
   (C5 ^a51 <F> ^a52 <C> ^a53 T4)
   (C6 ^a61 <B> ^a62 T5 ^a63 <A>)
   → actions)
```

ここで、r はルール No.、C1,C2,…,C6 はクラス名、$\hat{}a_{11}$,$\hat{}a_{12}$,…,$\hat{}a_{63}$は属性名、A,B,C,D,E,F は変数名、T1,T2,…,T5 は定数である。

いま、次のような WME が追加されたとする。

(C1 ^a11 3 ^a12 4 ^a13 5)

このとき、イントラコンディションテストに成功し、変数 A,B,C にそれぞれ 3,4,5 が代入される。

```
begin
(1) input wme,z;
   t ← a class-No. of wme;
(2) detect a set of rules H with the same class-No. by y from TB;
(3) if z="deletion" then
(4)    begin
       delete wme in WM;
       delete wme in α-memory;
       delete instansiation includeing wme in CS-memory;
(5)     for R∈H do
        begin
          intracondition test of a condition element E* with (6) class-No. t;
          if wme does'nt satisfy E* then go to L0;
          i ← rule-No. of R;
          j ← an ordinal number of e* in R;
          V← variable set in E*;
          for x∈V do
            if A_ijx(a_x)≠0 then A_ijx(a_x) ← A_ijx(a_x) − 1
            attribute-value-No. a_x in wme assigns to variable x
    L0: end
    end
  else z="addition"
(7)    begin
       add wme in WM;
       for R∈H do;
        begin
          intracondition test of a condition element E* with a class-No. t;
          if wme doesn't satisfy E* then fo to L1;
          i ← rulw -No. of R;
          j ← an ordinal number of E* in R;
(8)       add wme in α-memory for E*;
(9)       copy each attribute-value-No. of wme to the corresponding
          variable of E*;
          Thie operation assigns to variable x attribute-value-No. a_x
          V← variable set in E*;
          for x∈V do
            A_ijx(a_x) ← A_ijx(a_x) + 1;
(10)    COMBINATORICS*(R, j,wme)
    L1: end
    end
end
```

図3　提案2のアルゴリズム

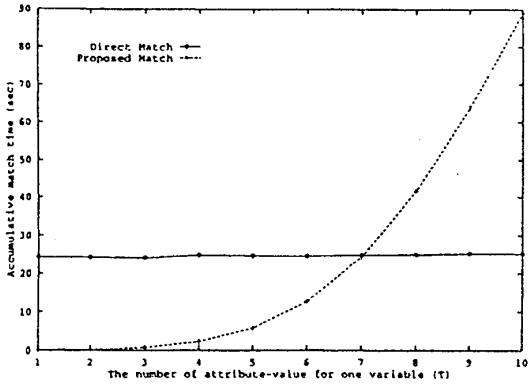

図4　ケース1の場合

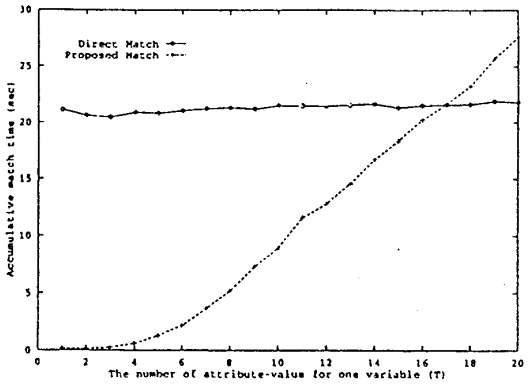

図5　ケース2の場合

PRECHECK*(1,L) は、このとき、$\hat{}a_{63}$の属性値に3、$\hat{}a_{61}$の属性値に4、$\hat{}a_{22}$,$\hat{}a_{52}$に5が入るWMEが存在するかどうかを配列A_{itx}で調べる。

2番目の条件要素に対応するαメモリに次のようなWMEが入っていたとする。

(C2 $\hat{}a_{21}$ 3 $\hat{}a_{22}$ 2 $\hat{}a_{23}$ T1)
(C2 $\hat{}a_{21}$ 4 $\hat{}a_{22}$ 1 $\hat{}a_{23}$ T1)
(C2 $\hat{}a_{21}$ 4 $\hat{}a_{22}$ 2 $\hat{}a_{23}$ T1)
(C2 $\hat{}a_{21}$ 3 $\hat{}a_{22}$ 1 $\hat{}a_{23}$ T1)
(C2 $\hat{}a_{21}$ 3 $\hat{}a_{22}$ 3 $\hat{}a_{23}$ T1)

このとき、配列A_{r2C}の値は次のようになる。

$$A_{r2C}(1) = 2, A_{r2C}(2) = 2, A_{r2C}(3) = 1,$$
$$A_{r2C}(4) = 0, A_{r2C}(5) = 0, \cdots\cdots$$

それ故、$A_{r2C}(5) = 0$から、2番目の条件要素を満足させるWMEが存在しないことが分かる。

4　実験

提案1と提案2の有効性を実験により示す。尚、計測に用いた計算機はSUN Microsystems Computer社のSPARC station ELCである。

4.1　提案1の実験

1つの条件要素に対する1000回の条件照合の累積時間を測定し、文献(6)の条件照合アルゴリズムと提案アルゴリズムを比較した。ただし、単一の条件要素の照合に絞った理由は、その他の処理が文献(6)とほとんど同じだからである。また、1000回の累積時間を求める理由は、1回あたりの条件照合時間が小さすぎて、使用したタイマでは正確に計測できないからである。

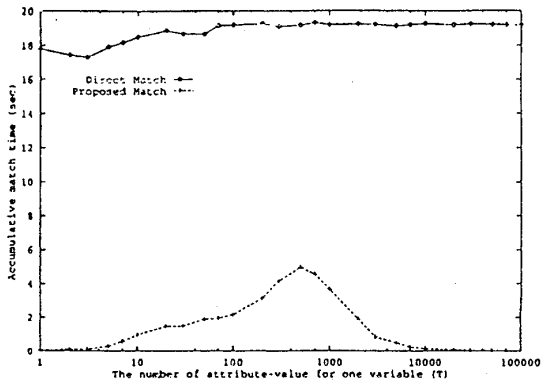

図6　ケース3の場合

4.1.1　実験方法

本実験で対象とする条件要素は、

(C$_i$ $\hat{}a_{i1}$ <X> $\hat{}a_{i2}$ <Y> $\hat{}a_{i3}$ <Z>)

のような3つの属性をもち、属性値にはすべて変数が充てられているものとする。

まず、条件要素を満足させるWMEを乱数で1000個作成し、それをαメモリの内容とする。具体的な作り方は、各変数に対して整数乱数を発生し、それをTで割ったときの余りを属性値とする。ここで、Tは属性値の取り得る値の数であり、各変数とも同じ値とす

る。

1つの条件要素に対して、条件要素を満足させるWMEをαメモリからすべて導く操作(条件照合)を1000回繰り返して、その累積時間を測定する。ただし、対象とする条件要素の変数の値の確定状況によって、次の3つの場合を考える。

(1) ケース1:変数の値が3つとも確定していない場合
(2) ケース2:変数の値が1つだけ確定している場合
(3) ケース3:変数の値が2つ確定している場合

4.1.2 実験結果

属性値の取り得る値の数Tを変化させながら、1つの条件要素に対する1000回の条件照合の累積時間を、文献(6)の条件照合アルゴリズムと提案アルゴリズムについて測定した。実験の結果をケース1,2,3の場合に分けて、それぞれ図4,5,6に示す。

実験結果から、Tが小さいときにはいずれも提案方式が勝っていることが分かる。Tが小さいと、条件照合の成功率が高くなり、高コストルールになりやすくなる。文献(6)の条件照合アルゴリズムはTの値に無関係なので累積時間は一定である(多少のぶれは乱数を用いていることによる)。それに対し、提案アルゴリズムではTの値が小さいうちは候補のWMEも少なく、計算コストが小さい。また、Tがある程度大きくなると逆に計算コストが小さくなる理由は、確定した変数の値を属性値にもつWMEが存在するかどうかを初めに調べることにより無駄な照合を省くことができたからである。

4.2 提案2の実験

4.2.1 実験方法

本実験では、Reteアルゴリズム、直接条件照合アルゴリズム、そして提案方法を直接条件照合アルゴリズムに加えた提案アルゴリズムの3つのアルゴリズムの処理時間を比較する。用いたルールは例2のルールである。

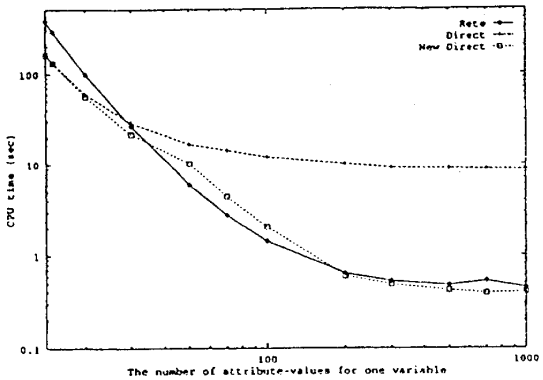

図7 WMEの追加における実験結果

各変数の取り得る値の数を同じにし、変数の取り得る値の数を変化させることにより、条件照合の計算コストを変化させる。変数の値や追加されるWMEのクラス名No.、属性値No.は一様乱数を用いて決める。但し、クラス名No.は6以下であり、変数の値や属性値No.は変数の取り得る値の数以下である。

実験は変数の取り得る値の数を変化させ、1000個のWMEを追加するのに要する累積時間、及び追加したすべてのWMEを削除するのに要する累積時間を測定する。

4.2.2 実験結果

WMEの追加に対する実験結果を図7に示す。一般に、変数の取り得る値の数が少なければ少ない程、条件照合の計算コストは大きくなり、高コストルールとなる。それ故、高コストルールに対しては、図7から提案アルゴリズムが最も効率が良いことがわかる。これは、提案アルゴリズムが直接条件照合アルゴリズムに、条件照合に失敗するかどうかをいち早く識別する機能を付加したからである。高コストルールでは、Reteアルゴリズムよりも直接条件照合アルゴリズムの方が、効率が良いことが知られている(6)。しかし、高コストルール以外では、極端に悪化してしまうことが欠点であった。本方式を用いれば、変数の取り得る値の数が増えれば増えるほど、直接条件照合アルゴリズムよりも効率が良くなってくる。この理由は、条件照合に失敗するケースが増えてくるからである。

WMEの削除に対する実験結果を図8に示す。提案アルゴリズムでは、削除処理に対する改善をして

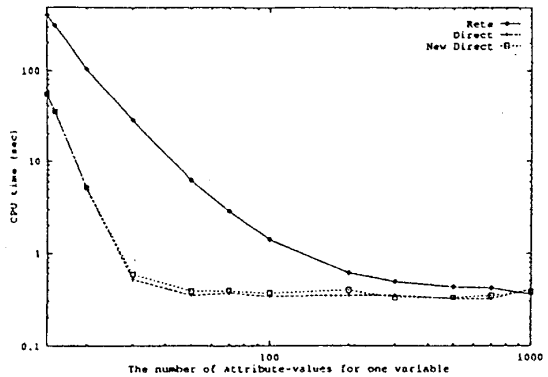

図 8　WME の削除における実験結果

いないので、直接条件照合アルゴリズムとほとんど同じ計算コストである。異なる点は、WME の削除により、配列 A_{ijx} の度数を減らす操作の手間が加算されるだけである。

以上のことから、提案アルゴリズムが直接条件照合アルゴリズムよりも勝っていることがわかる。

5　むすび

文献 (6) の条件照合アルゴリズムと提案 1 の条件照合アルゴリズムの主な違いは、1 つの条件要素あたりの条件照合であるので、実験結果から提案 1 の条件照合アルゴリズムが文献 (6) の条件照合アルゴリズムよりも優れた高コストルール対処法となることが分かる。今後の課題は、ケース 1 の場合でも効率がよくなるように改善することである。

提案 2 では、低コストのときに効率が悪い直接条件照合アルゴリズムの欠点を、条件照合に失敗するかどうかをいち早く識別する機能を付加することにより克服した。今後の課題としては、低コストルールに対しても、Rete より効率の良いアルゴリズムを構築することである。

謝辞 本研究を進めるうえで有益な御助言を頂いた富山大学工学部広瀬貞樹先生に深謝します。

参考文献

[1] Tambe,M. and Rosenbloom,P.:"Eliminatingn Expensive Chunks by Restricting Expressiveness ",IJCAI-89,pp.731-737(1989).

[2] 荒屋眞二，内山健次郎，高野啓，辻秀一:" 競合解消の分散化によるプロダクションシステムの高速実行方式" ，電気関係学会九州支部第 42 回連合大会 (1989).

[3] Miranker,D.P.,Brant,D.A.,Lofaso,B. and Gadbois,D.:"On the Performance of Lazy Matching in Production Systems",AAAI-90,pp.685-692(1990).

[4] 荒屋眞二，百原武敏:" プロダクションルールの分解とその効果" ，人工知能学会誌，7,1,pp.117-129(1992).

[5] 木村春彦:" プロダクションシステムのインスタンシエーション表現に関する一考察" ，人工知能学会誌，8,1,pp.91-101(1993).

[6] 木村春彦，住吉一之，小林真也，武部幹:" プロダクションシステムの直接条件照合アルゴリズム" ，電子情報通信学会論文誌，J77-D-II,2,370~379(1994).

[7] 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).

[8] Brownston,L.,Farrell,R.,Kant,E. and Martin,N.: "Programming Expert System in OPS5: An Introduction to Rule-Based Programming",Addison-Wesley(1985).