知識工学と人工知能 56 － 7
（１９８８．１．１３）

プロダクションシステムの トポロジカル・オプティマイザ

石田 亨　　横尾 真
（ＮＴＴ 情報通信処理研究所）

あらまし

ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑはﾜｰｷﾝｸﾞﾒﾓﾘ中のﾃﾞｰﾀ量が増大すると急激に性能が劣化することが知られている．これは，①ﾙｰﾙの条件判定に含まれる一連の結合演算(join)に要する時間が，通常の処理系ではﾃﾞｰﾀ量の２乗に比例すること，②不適切な条件の記述により，多量の中間的な演算結果が生成され，後続する結合演算の処理量をさらに増大させることが原因と考えられる．

本論文ではﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑの動作特性を表す統計ﾃﾞｰﾀを基に，結合演算の順序や組合せ方法(joinﾄﾎﾟﾛｼﾞｰ)を変更し，結合演算量を最小化するｱﾙｺﾞﾘｽﾞﾑを示す．また，このｱﾙｺﾞﾘｽﾞﾑに基づきﾄﾎﾟﾛｼﾞｶﾙ・ｵﾌﾟﾃｨﾏｲｻﾞを試作し，既存のﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑﾌﾟﾛｸﾞﾗﾑに適用した結果，人手による改善を凌ぐ効果が得られたので報告する．

"A Topological Optimizer for Production System Programs"

Toru Ishida and Makoto Yokoo
(NTT Communications and Information Processing Laboratories)

ABSTRACT

The efficiency of production systems rapidly decreases when the number of working memory elements becomes larger. This is because, in most implementations, the cost of join operations is directly proportional to the square of the number of working memory elements.

This paper describes an algorithm which minimizes the total cost of join operations in production system programs by selecting optimal join topologies based on their execution statistics. The result of implementing a topological optimizer and applying it to existing programs demonstrates that the optimizer generates more efficient programs than what obtained by manual optimization.

1．まえがき

ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑは，ﾜｰｷﾝｸﾞﾒﾓﾘ量が増大すると急激に性能が劣化する．これは，①ﾙｰﾙの条件判定に含まれる一連の結合演算(join)に要する時間が，通常の処理系ではﾃﾞｰﾀ量の２乗に比例すること*，②不適切な条件の記述により，多量の中間的な演算結果が生成され，後続する結合演算の処理量をさらに増大させることが原因と考えられる．この問題を解決するためにYES/OPS[9]，ART[1]等では，OPS5[2]を拡張し，利用者がﾙｰﾙの条件部で結合演算の順序や組合せ方法(joinﾄﾎﾟﾛｼﾞｰ)を自由に指定できるようにしている．

しかし，効率のよいjoinﾄﾎﾟﾛｼﾞｰをﾌﾟﾛｸﾞﾗﾑするのは容易なことではない．実際，最適なjoinﾄﾎﾟﾛｼﾞｰは，ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑﾌﾟﾛｸﾞﾗﾑの動作特性に依存するため，実行中に統計ﾃﾞｰﾀを測定し，それをもとに選択を行わねばならない．このことは，たとえﾙｰﾙは同一でも，対象とするﾃﾞｰﾀが異なれば（例えば交換機の故障診断ｼｽﾃﾑでは対象とする機種が異なれば）最適なjoinﾄﾎﾟﾛｼﾞｰが異なることを意味している．即ち，最適化はｴｷｽﾊﾟｰﾄｼｽﾃﾑ開発側だけの問題ではなく，運用側でも行う必要があるが，人手による最適化ではこの要求を満足することは不可能である．また，最適化されたﾌﾟﾛｸﾞﾗﾑはﾘｰﾀﾞﾋﾞﾘﾃｨが悪い．短期的な性能向上を得るために人手によりｿｰｽﾌﾟﾛｸﾞﾗﾑを変更し，長期的なﾒﾝﾃﾅﾋﾞﾘﾃｨを犠牲にすることには問題が多い．

そこで本論文では，人間の手を煩わすことなく高性能なﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑﾌﾟﾛｸﾞﾗﾑを生成することを目的に，結合演算の処理量が最小となるようjoinﾄﾎﾟﾛｼﾞｰを最適化するｱﾙｺﾞﾘｽﾞﾑを提案する．本ｱﾙｺﾞﾘｽﾞﾑは以下の特徴を有する．

① ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑの動作特性を表す統計ﾃﾞｰﾀをもとに最適化を行う．

② 複数のﾙｰﾙを一括して最適化し，結合演算をﾙｰﾙ間にまたがり可能な限り共通化する．

本ｱﾙｺﾞﾘｽﾞﾑは従来からの問合わせ最適化(query optimization)[7.8.10.11]の拡張である．しかし，ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑはﾌﾟﾛｸﾞﾗﾑであるため，ﾙｰﾙが（即ち問合わせが）繰り返し実行されることを考慮しなければならない点が異なっている．以下ではまずﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑのｺｽﾄﾓﾃﾞﾙを定義する．次に種々のjoinﾄﾎﾟﾛｼﾞｰを生成評価し，ｺｽﾄが最小のものを選択する最適化ｱﾙｺﾞﾘｽﾞﾑを提案する．指数ｵｰﾀﾞで存在するjoinﾄﾎﾟﾛｼﾞｰから，各種の制約を用いて可能性を削減するところにｱﾙｺﾞﾘｽﾞﾑの重点がある．

また本ｱﾙｺﾞﾘｽﾞﾑに基づき，実験用ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑPLANET[5]上にﾄﾎﾟﾛｼﾞｶﾙ・ｵﾌﾟﾃｨﾏｲｻﾞを試作した．既存のｴｷｽﾊﾟｰﾄｼｽﾃﾑ[6]の最適化に適用した結果，過去に行われた人手による最適化を凌ぐ効果を確認している．

2．トポロジカルな等価変換

図1～3にﾌﾟﾛﾀﾞｸｼｮﾝﾙｰﾙの例を示す．ﾙｰﾙはPLANETの構文則で記述している．先頭文字が？のものが変数であることを除くと，OPS5にほぼ等しい．andはまず内部のjoinを計算することを示している．図2は図1のrule1を最適化した結果である．ﾃﾞｰﾀ数が小さなｸﾗｽから順に条件が並び換えられている．図3は図1の2個のﾙｰﾙを一括して最適化した結果である．small-classとmiddle-classの結合演算を独立に切り出すことにより2個のﾙｰﾙ間での結合演算の共通化を図っている．

図4はrule1の取り得るjoinﾄﾎﾟﾛｼﾞｰを示している．RETEｱﾙｺﾞﾘｽﾞﾑ[3]が採用されているとすると，図4はRETEﾈｯﾄﾜｰｸのﾄﾎﾟﾛｼﾞｰを表していると考えることができる．以下では，ⓐⓑⓒ

OPS5を始めとする多くの処理系では，ﾃﾞｰﾀは単純なﾘｽﾄで管理され，結合演算はnested-loopで計算されている．ここでの記述はこの事実に基づいたものであるが，本論文で提案するｱﾙｺﾞﾘｽﾞﾑはﾃﾞｰﾀがﾊｯｼｭ[4]されている場合にも有効である．

```
(defrule rule1                (defrule rule2
  (context phase1)              (context phase2)
  (large-class   ?x)            (small-class   ?x)
  (middle-class ?x)             (middle-class ?x)
  (small-class   ?x)            -->
  -->                           (make ..... ))
  (make ..... ))
       (1)                           (2)
```

図1 プロダクションルールの例
Fig. 1 Example of production rules

```
(defrule rule1
  (context phase1)
  (small-class   ?x)
  (middle-class  ?x)
  (large-class   ?x)
  -->
  (make ..... ))
```

図2 ルール内最適化の例
Fig. 2 Example of intra-rule optimization

```
(defrule rule1                (defrule rule2
  (context phase1)              (context phase2)
  (large-class ?x)              (and (small-class  ?x)
  (and (small-class   ?x)            (middle-class ?x))
       (middle-class ?x))       -->
  -->                           (make ..... ))
  (make ..... ))
       (1)                           (2)
```

図3 ルール間最適化の例
Fig. 3 Example of inter-rule optimization

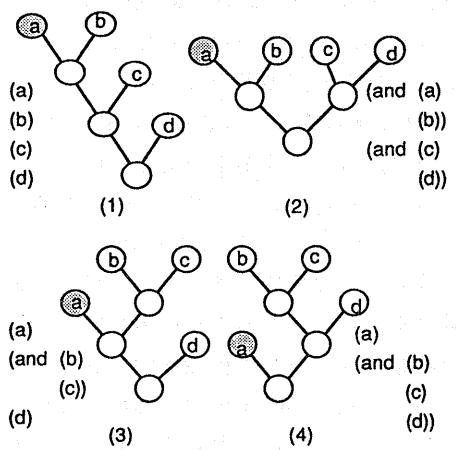


図4 種々のｊｏｉｎトポロジー
Fig. 4 Variations in join topologies

ⓓを1-input-testﾉｰﾄﾞ，それらの結合演算を行うﾉｰﾄﾞを2-input-testﾉｰﾄﾞと呼ぶ．

競合解決戦略としてMEAが用いられている場合にはⓐの位置は変えることができない．ⓑⓒⓓについては，どの様な順序になってもよいので，結局24通りの（一般には，指数ｵｰﾀﾞの）joinﾄﾎﾟﾛｼﾞｰが存在することになる．

3．コストモデル

3.1 ﾊﾟﾗﾒｰﾀ

図5に結合演算のｺｽﾄﾓﾃﾞﾙを示す．なお，以下の説明ではRETEﾈｯﾄﾜｰｸに関する知識を前提とする．ｺｽﾄﾓﾃﾞﾙは次に示すﾊﾟﾗﾒｰﾀから成る．

① T(oken)：各ﾉｰﾄﾞに到達したﾄｰｸﾝ（ﾜｰｷﾝｸﾞﾒﾓﾘの更新により生じる）の総数である．

② M(emory)：各ﾉｰﾄﾞのα-memory，β-memory（前ﾌﾟﾛﾀﾞｸｼｮﾝｻｲｸﾙでのselection，joinの計算結果がそれぞれ保存されている）に格納されたﾃﾞｰﾀの平均値である．

③ J(oin)：各ﾉｰﾄﾞで実行された2-input-test回数である．1個のﾄｰｸﾝとﾒﾓﾘ中の1個のﾃﾞｰﾀとの結合演算を1回の2-input-testと数えている．

④ C(ost)：各ﾉｰﾄﾞに至るまでに実行された2-input-test回数の総和である．

⑤ R(atio)：各ﾉｰﾄﾞでの2-input-testの成功確率である．

3.2 1-input-testﾉｰﾄﾞのﾊﾟﾗﾒｰﾀ

1-input-testﾉｰﾄﾞのﾊﾟﾗﾒｰﾀは，利用者が作成したﾙｰﾙを一度実行させて測定する．測定するﾊﾟﾗﾒｰﾀは以下の通りである．

(1) 1-input-testﾉｰﾄﾞでのToken

測定値はﾌﾟﾛｸﾞﾗﾑの動作特性にのみ依存し，joinﾄﾎﾟﾛｼﾞｰには依存しない．従って，どのようなjoinﾄﾎﾟﾛｼﾞｰをとる場合にも測定値をそのまま用いることができる．

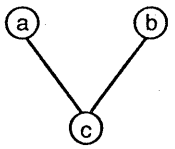
(2) 1-input-testﾉｰﾄﾞでのMemory

α-memory中のﾃﾞｰﾀ数の平均のとり方には様

<1-input-test node>

T(oken) : given
M(emory) : given
J(oin) : 0
C(ost) : 0
R(atio) : given

<2-input-test node>



When b is a positive condition element:	When b is a negative condition element:
Tc = (TaMb + TbMa) Rc	Tc = TaRc
Mc = MaMbRc	Mc = MaRc
Jc = TaMb + TbMa	Jc = TaMb + TbMa
Cc = Ca + Cb + Jc	Cc = Ca + Cb + Jc
Rc = Rb	Rc = Rb

図5 コストモデル
Fig. 5 Cost Model

様な方法が考えられる．ここではﾌﾟﾛﾀﾞｸｼｮﾝｻｲｸﾙ毎に，α-memoryに保存されているﾃﾞｰﾀ数を測定し，その平均値を取っている．この値もjoinﾄﾎﾟﾛｼﾞｰには依存しない．

(3) 1-input-testﾉｰﾄﾞでのRatio

1-input-testﾉｰﾄﾞのRatioとして，そのﾉｰﾄﾞを右入力とする2-input-testの成功確率を測定し，その値を設定する．この値は測定時のjoinﾄﾎﾟﾛｼﾞｰに依存する．従って，joinﾄﾎﾟﾛｼﾞｰが変化した時には，この値は近似値を与えることになる．

PLANETでは上記の計測ｵｰﾊﾞﾍｯﾄﾞは数%にすぎないため，計測は常時行っている．従って，任意の時点で，図6に示すような測定結果の表示を得ることができる．

3.3 2-input-testﾉｰﾄﾞのﾊﾟﾗﾒｰﾀ

2-input-testﾉｰﾄﾞ（図5ではⓒ）のｺｽﾄﾓﾃﾞﾙは右入力の1-input-testﾉｰﾄﾞ（図5ではⓑ）が，正の条件要素であるか，負の条件要素(not)であるかによって異なってくる．そこで，図5にはそれぞれの場合の計算ﾓﾃﾞﾙを提示している．

```
one   (context phase1)
  |     token: 1   memory: 0.5
  |------ one   (large-class ?x)
  |             token: 8   memory: 4.0
! two   token: 8   memory: 2.0   join: 8
  |     * one   (small-class ?x)
  |       |     token: 4   memory: 2.0
  |       |----- * one   (middle-class ?x)
  |       |              token: 6   memory: 3.0
  |------ * two  token: 12   memory: 3.0   join: 24
 two    token: 24  memory: 3.0   join: 48
terminal  instantiation: 24
```

one: 1-input-test node
two: 2-input-test node
! : 2-input-test with no join variables
* : shared node

図6 動作特性データの表示例
Fig. 6 Example of displaying statistics

(1) 2-input-testﾉｰﾄﾞでのToken，Join

Tokenが左方向から入力される場合には，Token数(Ta)と右ﾒﾓﾘ内のﾃﾞｰﾀ数(Mb)を掛け合わせた数(TaMb)の2-input-testが行われる．Tokenが右入力の場合にはその反対(TbMa)である．両者を加えたもの(TaMb+TbMa)が，このﾉｰﾄﾞで実行される2-input-test回数(Jc)である．

ﾃｽﾄの結果生成されるToken数(Tc)は，右入力が正の条件要素の場合には2-input-test回数にﾃｽﾄの成功確率(Rc)を掛けた値(JcRc)となる．一方，負の条件要素の場合には，左入力のﾄｰｸﾝ(Ta)がﾌｨﾙﾀされた値(TaRc)となる．

(2) 2-input-testﾉｰﾄﾞでのMemory

平均的なﾃﾞｰﾀ数(Mc)は，右入力が正の条件要素の場合には，左右のﾉｰﾄﾞのﾒﾓﾘ内の平均ﾃﾞｰﾀ数を掛け合わせた値(MaMb)に2-input-testの成功確率(Rc)を掛けたもの(MaMbRc)と考える．一方，負の条件要素を右入力とする場合には，左入力のﾒﾓﾘ(Ma)をﾌｨﾙﾀした結果の値(MaRc)とする．

```
clear R(ule-list);
push all rules to R;
sort R in the decreasing order of costs;
for r = 1st to the last rule of R;
  clear N(ode-list);
  push all 1-input-test nodes of r to N ;
  let k be a number of 1-input-test nodes;
  append pre-calculated 2-input-test nodes to N ;
  for i =2nd to the last node of N;
    for j =1st to the i-1th node of N;
      if all constraint conditions are satisfied
      then do;
        let n be a 2-input-test node created from
          i and j ;
        calculate papameters of n ;
        push n just after max(i, k)-th node of N
      end
    rof
  rof
  find the lowest-cost terminal node;
  construct an optimized version of r
rof
```

図7 最適化アルゴリズムの概要

Fig. 7 Outline of the optimization algorithm

(3) 2-input-testﾉｰﾄﾞでのCost

左右のﾉｰﾄﾞのｺｽﾄ(Ca,Cb)に自ﾉｰﾄﾞの計算ｺｽﾄを加えたものである．2-input-testの計算ｺｽﾄは一般にA*Jc+B*Tcで表すことができる．ここで，A,Bは適当な定数である．本論文ではとりあえず2-input-test回数の総和を最小化することに目標を置き，Cc=Ca+Cb+Jc（即ち，A=1,B=0）としている．しかし，ﾃﾞｰﾀがﾊｯｼｭされていれば[4]むしろ2-input-testによって生成されるﾄｰｸﾝ数が問題となるであろう．このような場合には，例えばCc＝Ca+Cb+Tcとするなど，処理系に合わせてｺｽﾄの計算式を適切に設定することが必要である．

(4) 2-input-testﾉｰﾄﾞでのRatio

2-input-testの成功確率(Rc)には，近似値として，右入力の1-input-testﾉｰﾄﾞのRatio(Rb)を用いる．

4．最適化アルゴリズム

4.1 アルゴリズムの概要

最適化ｱﾙｺﾞﾘｽﾞﾑは基本的には可能なjoinﾄﾎﾟﾛｼﾞｰを生成し，ｺｽﾄが最小のものを選ぶというものである．しかしながら，既に述べたように指数ｵｰﾀﾞのjoinﾄﾎﾟﾛｼﾞｰが存在するため，単純な生成・評価手法(generate and test)でこの問題を解くことはできない．一方，ｺｽﾄを構成するﾊﾟﾗﾒｰﾀが何種もあるため，従来の間合わせ最適化のように，特定のﾋｭｰﾘｽﾃｨｸｽを用いて準最適なﾄﾎﾟﾛｼﾞｰを生成することは困難である．

そこで以下では，生成・評価手法をﾍﾞｰｽに，生成されるjoinﾄﾎﾟﾛｼﾞｰを種々の制約を用いて削減するｱﾌﾟﾛｰﾁをとる．ｱﾙｺﾞﾘｽﾞﾑの概略を図7に示す．要点は以下の通りである．

① ﾙｰﾙは事前に実行させて計測したｺｽﾄ(2-input-testの総数）が最も大きなものから最適化する．これはﾙｰﾙ間にまたがる最適化で，最大の自由度をｺｽﾄの高いﾙｰﾙに保証するための措置である．ﾙｰﾙ間の最適化については4.3節で詳述する．

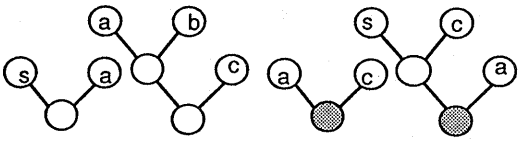
② 各ﾙｰﾙの最適化では，1-input-testﾉｰﾄﾞと計算済みのﾉｰﾄﾞ(pre-calculated 2-input-test node：詳細後述)を基に，それらのjoinを行うﾉｰﾄﾞを次々に生成する．新しく生成されたﾉｰﾄﾞが優先的に次のjoinの対象となる．これは，ﾙｰﾙの条件部を構成するjoinﾄﾎﾟﾛｼﾞｰを可能なかぎり早期に生成するための措置である．指数ｵｰﾀﾞで存在するjoinﾄﾎﾟﾛｼﾞｰを削減するための制約については4.2節で述べる．

③ 2-input-testﾉｰﾄﾞの生成が終了した時点で，ﾙｰﾙの条件部全体を構成するjoinﾄﾎﾟﾛｼﾞｰの内，最もｺｽﾄの低いものを選択し，最適化後のﾙｰﾙとする．

4.2 ｊｏｉｎトポロジー削減のための制約

joinﾄﾎﾟﾛｼﾞｰ削減のための制約としては以下のものを用いている．

```
(defrule example
  (context phase1) -- (s)
  (class-a ?x ?y) ---- (a)
  (class-b ?y ?z) ---- (b)
  (class-c ?z ?w) ---- (c)
  -->
  (make ..... ))
```



(1) possible join　　(2) unallowed join

図8 連結制約の例

Fig. 8 Example of connectivity constraint

(1) ｺｽﾄ制約

既に生成されているjoinﾄﾎﾟﾛｼﾞｰよりｺｽﾄの高いﾄﾎﾟﾛｼﾞｰを生成しないための制約である．ﾉｰﾄﾞnに至るまでにjoinされた1-input-testﾉｰﾄﾞの集合を1-input(n)で表す．もしnが1-input-testﾉｰﾄﾞであれば1-input(n)＝{n}とする．いま，ﾉｰﾄﾞnに対して，

$\exists m$ s.t. 1-input(n)$\subseteq$1-input(m) &
Cn$\geq$Cm

であればﾉｰﾄﾞnは生成しない．もし既に生成されているならば除去する．ｺｽﾄ制約によって最適解が得られなくなることはない．従って，ｺｽﾄ制約は本論文で提案する3種の制約の中で最も安全な制約である．

ｺｽﾄ制約を有効に生かすには，1-input-testﾉｰﾄﾞを多く含む低ｺｽﾄのjoinﾄﾎﾟﾛｼﾞｰを早期に生成すればよい．5章の評価では，最適化前のﾙｰﾙのjoinﾄﾎﾟﾛｼﾞｰ中に含まれるﾉｰﾄﾞを計算済みﾉｰﾄﾞ(pre-calculated 2-input-test node)として予め登録するようにしている．これによって，少なくとも最適化前の（即ち，利用者の記述した）ﾙｰﾙよりもｺｽﾄの高いjoinﾄﾎﾟﾛｼﾞｰの生成を防ぐことができる．種々のﾋｭｰﾘｽﾃｨｸｽ[1]を用いてｺｽﾄの低いjoinﾄﾎﾟﾛｼﾞｰを早期に求めることが今後の課題である．

(2) 連結制約

join変数が無い場合の2-input-test（以下，無条件2-input-testと呼ぶ）は多数の中間ﾃﾞｰﾀを生成する．連結制約は，避けられる無条件2-input-testを行わないための制約である．

ﾉｰﾄﾞn,mをjoinする際のjoin変数の集合をjoin-variable(n,m)で表す．いま，join-variable(n,m)＝ϕなるn,mに対して，

$\exists p,q \notin$ 1-input(n)$\cup$1-input(m)
s.t. join-variable(n,p)$\neq\phi$ &
join-variable(m,q)$\neq\phi$

であれば，nとmをjoinする2-input-testﾉｰﾄﾞは生成しない．

図8に連結制約の例を示す．連結制約はⓢとⓐの結合演算を妨げない．（この点が従来のconnectivity heuristics[10]と異なる．）ⓢとⓐは無条件2-input-testであるが，遅かれ早かれⓢと他のﾉｰﾄﾞの間で無条件2-input-testを行わざるを得ないからである．一方，ⓐとⓒの無条件2-input-testは制約される．なぜなら，ⓑとの結合演算を先に行うことによって，ⓐとⓒの無条件2-input-testを避けられる可能性があるからである．

(3) 順位制約

TokenやMemoryが大きな条件から結合演算が行われることを防ぐための制約である．1-input-testﾉｰﾄﾞpがqより優先順位が高いことをp$\gg$qで表す．（5章では，Token＊Memoryの小さなものほど高い優先度を与えた．）いま，ﾉｰﾄﾞn,mに対して，

$\exists p \gg q \in$ 1-input(m)
s.t. $p \notin$ 1-input(n)$\cup$1-input(m) &
join-variable(n,p)＝
join-variable(n,q)

であれば，n,mをjoinする2-input-testﾉｰﾄﾞは生成しない．なお，連結制約と順位制約は必ずしも最適解を保証しないことに注意する必要がある．

4.3 ルール間の最適化

複数のﾙｰﾙによって結合演算を共有できれば，各ﾙｰﾙ当りの演算ｺｽﾄは低下する．そこで，ﾙｰﾙ間の共有を促進するために以下の方法をとっている．

① 新たに生成する2-input-testﾉｰﾄﾞは，そのﾉｰﾄﾞを含みうる全てのﾙｰﾙで共有されると仮定する．即ちﾉｰﾄﾞのｺｽﾄは，本来のｺｽﾄを，今後最適化されるﾙｰﾙの内でそのﾉｰﾄﾞを利用し得るﾙｰﾙの数で割ったものとする．

② 他のﾙｰﾙで作成された2-input-testﾉｰﾄﾞはｺｽﾄ0で利用できることを仮定する．即ち，既に最適化されたﾙｰﾙに含まれる2-input-testﾉｰﾄﾞの内，利用し得るﾉｰﾄﾞを計算済みﾉｰﾄﾞ(pre-caluculated 2-input-test node)として登録する．この時，計算済みﾉｰﾄﾞのｺｽﾄには0を設定する．

5．試作・評価

4章で述べた最適化ｱﾙｺﾞﾘｽﾞﾑを適用して，PLANET上にﾄﾎﾟﾛｼﾞｶﾙ・ｵﾌﾟﾃｨﾏｲｻﾞを試作した．ｵﾌﾟﾃｨﾏｲｻﾞは利用者の記述したﾌﾟﾛｸﾞﾗﾑと動作特性を入力とし，最適化されたﾌﾟﾛｸﾞﾗﾑを出力する．このﾄﾎﾟﾛｼﾞｶﾙ・ｵﾌﾟﾃｨﾏｲｻﾞを既存のｴｷｽﾊﾟｰﾄｼｽﾃﾑの最適化に適用した結果を表1に示す．評価対象のﾙｰﾙは，論理回路の最適化を行うｴｷｽﾊﾟｰﾄｼｽﾃﾑ[6]の中核を構成する33ﾙｰﾙである．論理回路としてはﾜｰｷﾝｸﾞﾒﾓﾘｴﾚﾒﾝﾄ数が300～400の比較的小規模なものを用いた．最適化の結果，2-input-testの総数が1/3に，CPU性能が約2倍に向上した．なお，このｴｷｽﾊﾟｰﾄｼｽﾃﾑは本研究が始まる以前に開発者自身の手によって，最適化が行われている．特筆すべきことは，ﾄﾎﾟﾛｼﾞｶﾙ・ｵﾌﾟﾃｨﾏｲｻﾞがｴｷｽﾊﾟｰﾄｼｽﾃﾑ開発者自身の手による性能改善を凌ぐ効果を導き出していることである．

一方，最適化に要する処理時間は展開ﾉｰﾄﾞ数の2乗に比例する．評価対象のｴｷｽﾊﾟｰﾄｼｽﾃﾑには，条件数が20を越える大型のﾙｰﾙが多数存在するため，これらの制約を用いても現在最適化に約20分を要している．処理系の性能を上げると共に，特に展開ﾉｰﾄﾞ数が多くなるﾙｰﾙについては，最適化を途中で切り上げるなどの現実的な解決策をとることが必要である．

表1 最適化の効果

ﾙｰﾙ番号	条件要素	2-input-test回数			展開ﾉｰﾄﾞ数
		最適化前	人手改善後	最適化後	
1	21	46432	47888	22396	179
2	18	29548	27244	494	198
3	17	29548	27244	494	92
4	22	25513	7813	144	236
5	21	25513	7813	144	94
6	18	10322	3749	3749	552
7	17	10322	3749	3749	194
* 8	15	9966	9966	12539	228
9	7	9830	1180	1180	99
10	6	9278	630	630	20
11	17	8566	8566	4640	116
12	7	7656	520	760	44
13	6	7544	408	648	22
* 14	23	3160	4616	13780	76
15	11	1918	1918	1865	119
16	20	1336	1336	1325	413
：	：	：	：	：	：
：	：	：	：	：	：
計33	平均	241329	159517	75002	平均
ﾙｰﾙ	14.2	(1.00)	(0.69)	(0.53)	131.8

・()内はPLANETでのCPU時間比．
・ﾙｰﾙ間で共有されているtestの回数は，共有されているﾙｰﾙ数で割った後，加算している．
・*では最適化後のtest回数が最適化前を上回っている．しかし必ずしも最適化に失敗している訳ではない．例えばﾙｰﾙ1と14は多くのtestを共有している．2ﾙｰﾙのtest回数の和を比較すると，むしろ最適化効果が現れていることが分かる．

6．他の研究との関連

これまでに，ﾃﾞｰﾀﾍﾞｰｽの問合わせ最適化に関する研究が数多く報告されている．joinﾄﾎﾟﾛｼﾞｰの最適化に関連する研究としては，以下のものがある．

①Jarke[7]は多数の問い合わせを一括して最適化するMultiple Query Optimizationを提案している．これは本論文で提案する多数のﾙｰﾙの一括最適化とよく似た方式である．

②Kim[8]はさらに，ﾌﾟﾛｸﾞﾗﾑ中で類似の問い合わせが繰り返し実行される場合に，それらを一括して最適化するGlobal Query Optimizationを提案している．ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑの動作は

```
LOOP;
  Q1; ----  ﾙｰﾙ1の条件照合
  Q2; ----  ﾙｰﾙ2の条件照合
   :
  Update database
END
```

で近似できるので，Global Query Optimizationはﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑの最適化に極めて近い問題であるということができる．

しかしこれらの研究は，同時に実行可能な問合わせの最適化が目的であり，ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑのように既に行われた問合わせの結果を保存し再利用するという発想は生まれてこない．ﾃﾞｰﾀﾍﾞｰｽでは最適化にﾃﾞｰﾀ特性（ﾃﾞｰﾀ数）を用いるが，ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑの最適化ではそれに加えてﾌﾟﾛｸﾞﾗﾑの動作特性（ﾃﾞｰﾀ更新回数）を用いるのはこの相違に基づくものである．

問題解決の分野ではSmithやWarren[10,11]等によって，条件の連接により構成される問合わせの最適化が研究されている．ﾃﾞｰﾀﾍﾞｰｽの最適化同様，ﾌﾟﾛｸﾞﾗﾑの動作特性が考慮されていない点がﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑの最適化と異なっている．

7．むすび

ﾌﾟﾛﾀﾞｸｼｮﾝﾙｰﾙの条件部のjoinﾄﾎﾟﾛｼﾞｰを最適化するｱﾙｺﾞﾘｽﾞﾑとその効果について述べた．評価対象としたｴｷｽﾊﾟｰﾄｼｽﾃﾑは比較的小規模なものであるが，結合演算のｺｽﾄが1/3に削減されるという大きな効果が得られた．一般にﾙｰﾙ数，ﾃﾞｰﾀ数が増えれば最適化の効果は増大する．従って，ｴｷｽﾊﾟｰﾄｼｽﾃﾑの実用化が進むにつれﾄﾎﾟﾛｼﾞｶﾙ・ｵﾌﾟﾃｨﾏｲｻﾞはその有効性を増すものと考えている．

謝辞

日頃御指導戴くNTT情報通信処理研究所吉田清主幹研究員，評価用ﾌﾟﾛｸﾞﾗﾑを提供して戴いた石川雄三主幹研究員，並びに討論戴いた方々に感謝致します．また，ﾃﾞｰﾀﾍﾞｰｽの立場からｺﾒﾝﾄを戴いた神戸大学田中克己助教授，京都産業大学吉川正俊博士に感謝します．

参考文献

(1) Clayton, B. D., "ART Programming Tutorial", Inference Corp. (1987).

(2) Forgy, C. L., "OPS5 User's Manual", CMU-CS-81-135 (1981).

(3) Forgy, C.L., "Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem", Artificial Intelligence, Vol.19, pp. 17-37 (1982).

(4) Gupta, A., C. L. Forgy, D. Kalp, A. Newell and M.Tambe,"Results of Parallel Implementation of OPS5 on the Encore Multiprocessor", CMU-CS-87-146 (1987).

(5) 石田　亨，"ﾃﾞｰﾀ駆動型ﾌﾟﾛﾀﾞｸｼｮﾝｼｽﾃﾑによる意味ﾈｯﾄﾜｰｸの探索"，情報処理学会論文誌，Vol.28, No.10 (1987).

(6) 石川，仲西，中村，"論理ｹﾞｰﾄ最適化ｴｷｽﾊﾟｰﾄｼｽﾃﾑについて"，情報処理学会第34回全国大会 (1987).

(7) Jarke,M., "Common Subexpression Isolation in Multiple Query Optimization", in Query Processing in Database Systems, Springer (1984).

(8) Kim, W., "Global Optimization of Relational Queries: A First Step", in Query Processing in Database Systems, Springer (1984).

(9) Schor, M.I., T.P.Daly, H.S. Lee and B.R.Tibbitts, "Advances in RETE Pattern Matching", AAAI-86 (1986).

(10) Smith, D. E. and M. R. Genesereth, "Ordering Conjunctive Queries", Artificial Intelligence,26,pp.171-215 (1985).

(11) Warren, D.H.D., "Efficient Processing of Interactive Relational Database Queries Expressed in Logic", 7th VLDB (1981).