遺伝的プログラミングによる排他制御プログラムの生成

田村 真司† 宝珍 輝尚‡ 野宮 浩揮‡

†京都工芸繊維大学 工芸科学部〒606-8585 京都市左京区松ヶ崎御所海道町

E-mail: †gurugoo.tail@gmail.com, ‡{hochin, nomiya}@kit.ac.jp

あらまし 本論文では，遺伝的プログラミング（GP）による排他制御プログラム生成システムの設計，実装，実験について報告する．ここでは，任意の特徴を持つトランザクションに対応した排他制御機構を生成するために，GP で使用するノードとして変数の値の変更といった一般性の高いものを採用することで，従来の排他制御プログラムの混合手法ではない新たな手法で環境に合ったトランザクション特徴に特化した排他制御プログラムの生成を模索する．変数の値の変更という細粒度のノードを使用するので，正常に動作するプログラムを生成するために突然変異における機能単位の導入や適合度計算の改良，遺伝的操作での大世代への対応といった対応を行った．

キーワード 排他制御プログラム，遺伝的プログラミング，自動生成

Generation of Concurrency Control Program by Using Genetic Programming

Shinji TAMURA† Teruhisa HOCHIN‡ and Hiroki NOMIYA‡

†School of Science and Technology, Kyoto Institute of Technology Goshokaido-cho, Matsugasaki, Sakyo-ku, Kyoto_shi, Kyoto 606-8585 Japan

E-mail: †gurugoo.tail@gmail.com, ‡{hochin, nomiya}@kit.ac.jp

Abstract This paper presents the design, the implementation, and the experiment of generation system of concurrency control program by using genetic programming. For generating concurrency control system according to the feature of transactions, a new method is proposed. This method adopts general operations such as changing variables as nodes used in genetic programming. This will bring us more general concurrency control program than the previous method, which has adopted parts of traditional concurrency control as nodes. For the purpose of generating appropriate program with fine-grained nodes, we introduce the function unit at the mutation, improve the fitness calculation, and enable the system execution in large generation.

Keyword Concurrency Control Program, Genetic Programming, Automatic Generation

1. はじめに

日進月歩で技術が進化している今日，様々なコンピュータが環境に合わせて発達し，それと同時にデータベースが多種多様な場面で必要となっている．しかし，様々な分野の要求に応えるには，単一の汎用的なデータベース管理システム（Database management system, DBMS）では対応できない部分が出てくる．このことから，各分野に対応した機能を持つ DBMS が望まれる[1]が，その構築は難しい．また，実現できたとしても特定の場面でしか使わない不要な機能が多くなる恐れがある．つまり，各分野に特化した DBMS が必要となるのであるが，その構築には多大なコストと時間が必要となる．このことから，DBMS の拡張性が強く求められている[2]．

本研究では，DBMS の性能や信頼性に関わる最も重要な機構の一つである排他制御機構の生成について検討する．データベースを利用する際には，分野によって異なるトランザクションが発生すると考えられる．それは例えば，データを読む操作がほぼ全てを占めていたり，ある一つのデータに対しての操作が集中したり，といったように様々な特徴があり，ある特徴に対しては適切な排他制御機構も別の特徴を持つ分野で運用すると著しく効率が落ちてしまうことがある．発生するトランザクションの特徴に応じて最適な排他制御機構を生成できるならば，利用分野に最適な DBMS の構築に繋がる．

水野らは，遺伝的プログラミングを用いてトランザクションの特徴に合った排他制御プログラムを生成する試みを行っている[3]．ここでは，二相ロック法を実現するプログラムと時刻印順序法を実現するプログラムをもとにして基本機能を抽出し，遺伝的プログラミングでプログラムを表現する木（プログラム木）のノードとしている．初期プログラム木として二相ロック法用のプログラム木を与えたところ，より効率的な処

理を行うプログラム木を得るという結果が得られており，トランザクションの特徴を反映した排他制御プログラムの生成という目的は達成されていると考えられる．しかしながら，プログラム木のノードは二相ロック法と時刻印順序法における基本機能であるため，これらをもとに遺伝的操作を施しても，これらの方法の混合手法が得られる程度で，全く新しい制御方法が得られる可能性は低い．

そこで本論文では，トランザクションの特徴に応じた排他制御プログラムの自動生成を目的として，水野の排他制御プログラム自動生成システムにおけるプログラム木のノードを汎用性の高いものとし，様々な制御プログラムの生成をも可能とする手法を提案する．提案する手法は，排他制御プログラムが各データならびに各トランザクションに対して変数を設定し操作していることを利用するもので，プログラム木のノードとして変数の設定操作を行うものを導入する方法である．

以降，2．では遺伝的プログラミングについて概説し，3．で前提とする排他制御プログラム自動生成システム[3]について述べる．4．でプログラム木のノードの提案等を行い，最後に 5．でまとめる．

2. 遺伝的プログラミング

遺伝的アルゴリズム（Genetic Algorithm， GA）とは進化論的な考え方に基づいてデータを操作し，最適化の問題や学習，推論を扱う手法である．GA のアルゴリズムは次のようになっている．

（１） 現世代の集団を生成する．
（２） 現世代の集団内の各個体に対して適合度を計算する．
（３） 適合度をもとに現世代の集団から個体を選択する．
（４） （３）で選択した個体に対して，突然変異・交差などの操作を行い，次世代を生成する．
（５） （２）～（４）を必要数繰り返し，全ての世代の中で最高の適合度を持つ個体を解として得る．

遺伝的プログラミング（Genetic Programming，GP）は GA を構造的な表現ができるように拡張したものである．GP における個体は木構造で表し，突然変異・交差などの遺伝的操作を部分木の変更によって実現させる．また，木構造はＳ式で表すこともできる．このときの対応関係を図 1 に示す．図 1 の F1，F2 は非終端記号，T1，T2 は終端記号を表している．

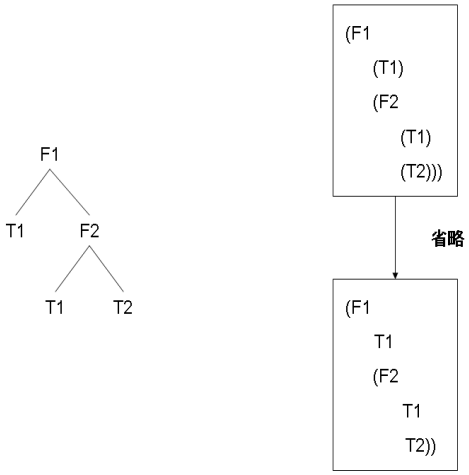

図 1．プログラムの木構造表現とＳ式表現

3. 排他制御プログラム生成システム

3.1. 排他制御プログラムの概要

本論文で前提とするシステム[3]のシステム構成を図 2 に示す．このシステムは以下の順序で実行される．

（１） トランザクションの特徴を設定して，必要な数のトランザクションを生成する．
（２） 各トランザクションからのオペレーションを生成順に並べ，スケジュールとして得る．
（３） 遺伝的操作により排他制御プログラムを得る．
（４） 排他制御プログラムで（２）で得たスケジュールを制御して適合度を得る．
（５） 最終世代に到っていなければ（４）で得た適合度をもとに（３）の操作に戻る．最終世代なら終了して最良個体を得る．

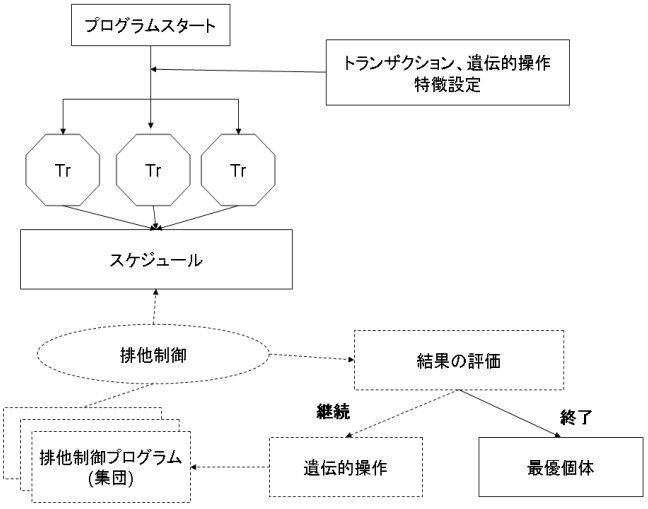

図 2．排他制御プログラム生成システムの概要

3.2. ノード

個体（プログラム）を構成するノードとして汎用的な二相ロック法や時刻印順序法などの代表的な排他制御を実現するプログラムを分解・整理したものを採用している．

ノードが代表的な排他制御プログラムの基本機能であるため，遺伝的操作を施しても，これらの方法の混合手法が得られる程度で，全く新しい制御方法が得られる可能性は低い．

3.3. 適合度の計算

適合度の計算に関わるものは大きく分けて 5 つあり，以下の順で計算する．

（１） 直列可能性の検証

直列可能性の検証とは，複数の処理を並行して行った時に，それらの処理を順番に行った時と同じ結果が得られるかどうかを検証することである．

制御済みスケジュールにおいて，直列可能性が保証される場合のみ以降の適合度計算を行う．直列可能性が保証されない場合は適合度を最低に設定する．

（２） 同時実行性の検証

この計算を行う時は，直列可能性が保証されているので，単純に一度に実行できるトランザクションの数が多ければ多いほど効率的になると考えてよい．

直列スケジュールの長さを l_0，制御後スケジュールにおける同時実行オペレーションの最大同時実行数を $c_{\max}$，同時実行を考慮した総ステップ数をスケジュールの長さで割ったものを平均同時実行数 c_{ave} として，式(1)により得られる値を同時実行性の評価値とする．

$$E_1 = l_0 \times c_{\max} \times c_{ave} \qquad (1)$$

（３） トランザクションの応答時間

トランザクションの集合 $\{T_1, T_2, \cdots, T_n\}$ があるとき，T_n の直列スケジュールにおける応答時間を ts_n，制御済みスケジュールにおける応答時間を tc_n とする．制御済みスケジュール中でアボートされたものを除いて式（2）の計算を行い，トランザクションの応答時間の評価値とする．

$$E_2 = \sum_{k=1}^{n} \frac{ts_k}{n \cdot tc_k} \qquad (2)$$

（４） トランザクションのアボート率

全トランザクションのうちアボートされた割合 p_{abort} を求め，式(3)の計算を行い，トランザクションのアボート率の評価値とする．

アボート率が 0.5 以上の時，性能が悪いことが容易に分かるため，2 乗している．

$$E_3 = \begin{cases} 1 - p_{abort} & if \quad p_{abort} < 0.5 \\ (1 - p_{abort})^2 & otherwise \end{cases} \qquad (3)$$

（５） プログラムサイズによる補正

プログラムサイズを考慮する計算を行う．

具体的にはS式プログラムの非終端記号を数えて N_f とし，プログラム起動前に設定しておいた，どれだけプログラムの大きさを重視するかの値(ここでは 0.005 としている）を乗算して得られた値をプログラムサイズの評価値とする．

$$E_4 = N_f \times 0.005 \qquad (4)$$

（６） 最終的な適合度

（1）～（5）までの計算を行い，最終的な適合度 E を求める．今までの結果をまとめると式(5)となる．

$$E = E_1 \times E_2 \times E_3 - E_4 \qquad (5)$$

4. 改良

4.1. ノードの一般化

より汎用的な排他制御プログラム作成システムとするために，プログラム木のノードをより一般的な基本機能のものとする．大野らは，排他制御法の拡張可能化を目指し，データに関する情報，トランザクションに関する情報，システムに関する情報，ならびに，演算の種別を個別に管理し，演算のパターンを含めたアクションを利用者が遷移図により記述する方式で実現を試みた[4]．ここで整理されているように，排他制御法はデータ，トランザクションやシステムの状態等を表す変数を使用して制御を行っている．

そこで，まず，データやトランザクションが個別に持つ変数を宣言・操作可能とする．この変数を不定変数と呼ぶ．

ここで，排他制御法では，データとそれを操作するトランザクションを関係付けて管理することも多い．

これをデータの不定変数とトランザクションの不定変数を用いて管理させようとすると，データとトランザクションの関係付けは多くの場合定型的な処理となるが，定型的な一連の操作を遺伝的プログラミングで生成するのは困難であり，また，多くの世代を必要とするので効率が悪い．そこで，データとトランザクション間で持つ特別な変数を導入する．この変数を共有不定変数と呼ぶ．

変数の操作で必要になるノードとして考えられるのは次の 4 つである．

（1） 変数の値を設定する．
（2） 変数の値を得る．
（3） 変数の値を 1 増減させる．
（4） 変数を消去する．

これらをトランザクションの不定変数に対する処理ノード，データの不定変数に対する処理ノード，共有不定変数に対する処理ノードの 3 種類作成した．

変数の操作は，対象になっているオペレーションが属するトランザクションや操作するデータに対して行うこととする．

また，変数の値に応じて処理を変更させる必要があるので条件判定ノードが必要となる．条件判定として採用したのは次の 6 つである．

（1） ある値と別の値が等しいか．
（2） ある値が別の値と比べて小さいか．
（3） トランザクションが不定変数を持っているか．
（4） データが不定変数を持っているか．
（5） 共有不定変数が存在するか．
（6） デッドロックに陥っていないか．

（6）のデッドロックとはすくみ状態のことである．デッドロックとは二相ロック法で発生する問題で，2 つ以上のトランザクションがロックの影響によって互いに他のトランザクションがロックを解放するのを待つ状態になることを指す．

ここでは操作対象のデータと共有不定変数で連なっているトランザクションの中に状態 WAIT があるかどうか調べて，存在すればデッドロックと判定するノードとなっている．

他にも，トランザクションの ID を得るノードや実数値を返してくれるノード，共有不定変数で連なるトランザクション全てに対して次の操作を行わせるようなノードを用いれば，代表的な排他制御法である二相ロック法や時刻印順序法を表すことができる．

4.2. 突然変異における機能単位の導入

変数の値の変更操作をノードとすることで，より自由な形のプログラムを得ることができるようになるはずである．しかし，変数の値変更をノードで行う場合，プログラムとしてみるととても細かく，正常に動作するプログラムになるには多くのノードを必要とすることが予想される．

多くのノードが必要となるプログラムで一部のみ突然変異を起こすことは，評価値を大幅に下げる可能性がある．また，多くのノードが必要となるプログラムを遺伝的操作で正常に動作可能とするには多くの世代を必要とし，さらに，そのようなプログラムが生成される確率は非常に低い．そこで，新たな木を乱数で生成するのではなく，突然変異が発生した際に事前に設定した確率で規定木と入れ替えることによりこれらの問題に対処することにする．

ここでは規定木として，4.1 のノードで作成した時刻印順序法，二相ロック法の一部を採用している．現在 6 種類で，時刻印順序法の READ 時の動作，時刻印順序法の WRITE 時の動作，二相ロック法のデッドロックの判定とその後の動作，二相ロック法の共有ロック時の動作，二相ロック法の排他ロック時の動作，ならびに，二相ロック法の共有ロック法から排他ロック法への変更時の動作である．

4.3. 適合度計算の改良

3.3. のシステムの適合度計算の（3）トランザクションの応答時間について考察したところ，個々のトランザクションの応答時間についての計算を行っているが，スケジュール全体にかかる時間に触れていない．そこで，式(2)を式(6)に変更することにする．

$$E_2 = \frac{1}{2n} \cdot \{(n - n_{abort}) \cdot \frac{ts}{tc} + \sum_{k=1}^{n} \frac{ts_k}{tc_k}\} \qquad (6)$$

ここで，n_{abort} はアボートしたトランザクションの数，ts は直列スケジュールにおけるスケジュール開始から終了までの応答時間，tc は制御済みスケジュールにおけるスケジュール開始から終了までの応答時間，他は式(2)と同じである．

これによってスケジュール全体の速さも考慮され，高速化したときの評価値がより大きくできる．

ここでの応答時間は，オペレーション READ, WRITE の時間とそのスケジュールを制御する時の不定変数を宣言する時間を足したものとする．実機を用いて評価を行い，READ，WRITE と不定変数の宣言を 10：20：1 とすることにした．

4.4. 競合等価の重視

排他制御プログラムで競合等価な制御済みスケジュールが得られない，ということがあっては問題なので，全てのテストスケジュールに対して制御済みスケジュールが競合等価であれば適合度を大きくすることとして，式（7）を採用した．

$$E_5 = \begin{cases} 1.5 & if \quad p_{not\_conflict\_equal} = 0 \\ 1 - p_{not\_conflict\_equal} & else \quad if \quad p_{not\_conflict\_equalt} < 0.5 \\ (1 - p_{not\_conflict\_equal})^2 & otherwise \end{cases} \tag{7}$$

ここで，$p_{not\_conflict\_equal}$ はテストスケジュールを操作して得た制御済みスケジュールが競合等価ではない割合である．結果として得られる適合度 E は式(5)から式(8)となる．

$$E = (E_1 \times E_2 \times E_3 - E_4) \times E_5 \tag{8}$$

4.5. 大世代対応

ノードの細目化により実機のメモリ限界である世代数 2000 では求めたい結果を得ることが難しいと思われた．そこで，2000 世代ごとに最終世代と最良個体を次の実行に持ち越すことで擬似的に大量の世代を動作可能とした．スケジュールは毎プログラム起動時に生成を行い，生成したスケジュールに対して，前々回起動時の最良個体，前回起動時の最良個体で操作を行い適合度を求めて，高いほうの最良個体をこのプログラム開始時の最良個体とすることで，特定のスケジュールに特化しないようにした．

5. 実験

5.1. 初期設定

排他制御プログラムの生成実験で使用するトランザクションの初期設定と遺伝的操作の初期設定をそれぞれ，表 1 と表 2 に示す．

ここでは，操作すべきデータ数 p2 をトランザクションの数 p1 より小さくすることで意図的に多数の競合が発生する設定とした．また，総世代数を 120 万として以降の実験を行った．

表 1　トランザクションの特徴設定

設定項目	内容	値
p1	同時に実行するトランザクションの数	4
p2	トランザクションが操作するデータの最大数	3
p3	命令（オペレーション）に WRITE が含まれる確率	0.4
p4	トランザクションが発するオペレーションの最大数	5
p5	トランザクションが途中で終了する確率	0.1

表 2　GP の初期設定

設定項目	内容	値
s1	プログラム毎の世代数	2000
s2	一世代における個体数（集団サイズ）	40
s3	初期世代になるプログラム指定の有無	実験により異なる
s4	プログラムの最大深さ	8
s5	プログラムの成長方法	ランダム
s6	非終端記号における交叉確率	0.1
s7	終端記号における交叉確率	0.6
s8	コピーのみを行う確率	0.1
s9	評価に用いるスケジュールの数	40
s10	小さいプログラムを重視する割合	0.005
s11	突然変異の際ランダムで木を生成する確率	0.5
s12	入れ替えのため事前に設定した木の数	6

5.2. 実験結果

初期世代を指定無しとして生成実験を行った．結果として得られたプログラムは 119 万 9224 世代目に得られたもので，後述のように単純なロック構造を用いたアルゴリズムであり，全てのテストスケジュールに対して競合等価であった．このとき，生成したプログラムで新たにテストスケジュール 100 個に対して評価を行ったところ，適合度は約 43.23 であった．このプログラムを簡単化した時の流れを図 3 で示す．また，図 3 で使用したノードについては，まとめて表 3 に示す．プログラムの解説を次に述べる．

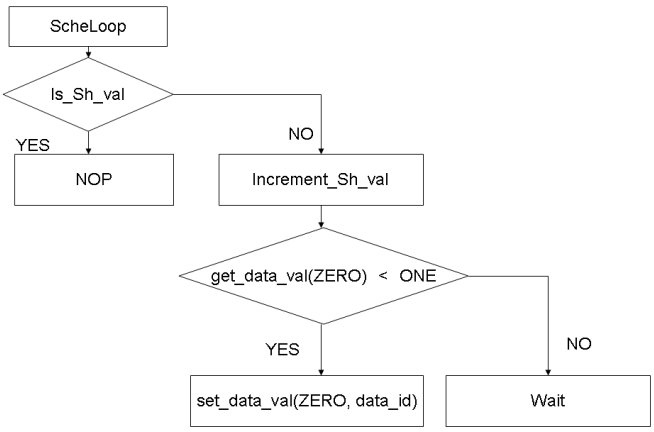

図 3. 初期世代指定無しプログラムのフロー

表 3. 図 3 で使用したノード

ノード	引数の数	動作	種別
ScheLoop	1	スケジュールの先頭から順に，全てに対して引数を実行する.	非終端
Is_Sh_val	2	現オペレーションが属するトランザクションと操作するデータ間の共有不定変数が存在すれば『引数 1』を実行し，存在しなければ『引数 2』を実行する.	非終端
NOP	0	操作なし	終端
Increment_Sh_val	0	現オペレーションが属するトランザクションと操作するデータ間の共有不定変数の value を 1 増やす. 宣言されていなければ value=1 の変数として宣言する.	終端
get_data_val	1	現オペレーションが操作するデータの id=『引数 1』の不定変数の value を得る. 存在しなければ 0 を得る.	非終端
ZERO	0	数値 0 を得る.	終端
ONE	0	数値 1 を得る.	終端
data_id	0	現オペレーションが操作するデータの id を得る.	終端
Wait	0	現オペレーションが操作するデータの共有不定変数の連なりを見て状態 Wait のもの以外を全て現オペレーションよりも前に移動する. また，自トランザクションの状態を Wait に変更する.	終端

まず，共有不定変数が存在するかを調べ(Is_Sh_val)，存在する場合は処理なし(NOP)，存在しない時は共有不定変数を値 1 で宣言(increment_Sh_val)して次の条件分岐を行う. ここではデータの不定変数 ID=1 の値が 1 より小さいかどうかを調べ(get_data_val(ZERO)<ONE)，1 より小さければデータの不定変数 ID=1 を値 data_id として宣言する(set_data_val(ZERO,data_id)). 1 以上であればトランザクションを待ち状態として後回しにする(Wait). これは，各データに対して最初に操作を行うトランザクションのみがデータの不定変数の値を変更し，それ以外のトランザクションは Wait を実行するという簡単なアルゴリズムということである.

同様の実験を初期木に時刻印順序法を指定して行ったところ，119 万 3539 世代目に最良個体が生成され，適合度は約 40.81 であった. アルゴリズムも初期木指定無しとほぼ同様の結果を得ることができた. これは大量の競合が発生する状況に適応するように進化した結果だと思われる. なお，初期木である時刻印順序法の適合度は約 21.30 であった.

一方，初期木に二相ロック法を指定した実験では，119 万 9678 世代目で最良個体を得られたが，プログラムの適合度は約 41.14 で，初期木指定無しから生成したプログラムとほぼ変わりなく，初期木の二相ロック法の適合度である約 65.22 と比較すると低下していた. 原因は，遺伝的操作中に特定のテストスケジュール集合に対して二相ロック法よりも良いプログラムができてしまい，最適な形が崩されたからではないかと思われる. テストスケジュールの数を増加させる等の対策が必要と考えられる.

また，表 1 の p3 の値を 0.1 として同様の実験を行ったところ，88 万 4068 世代目に最良個体を得ることができた. この条件での時刻印順序法の適合度が約 251.40，二相ロック法の適合度が約 289.90 であり，得られた最良個体の適合度が約 345.62 であった. この結果から，競合が頻繁に発生しない設定であれば二相ロック法よりも良質な排他制御プログラムが得られる可能性があることが分かった.

6. おわりに

本論文では，排他制御プログラムが各データならびに各トランザクションに対して変数を設定・操作していることを利用して，プログラム木のノードで変数の設定・操作を行わせるシステムの作成を試みた. この結果，今までにない形の排他制御プログラムを生成することはできなかったが，競合する条件で実験時には初期木を指定しなくても，ロックを使うプログラムを生成することができた.

今後は，現在は考慮していない各データの使用頻度

等を含めたトランザクションの生成による良質なスケジュールを用いて，他の初期条件を変更するとどのようなプログラムを得ることができるかについて実験を行っていく予定である．また，生成したプログラムを人が見て理解することはかなり困難である.そのため，生成したプログラムから意味のない部分を消去するなどして簡潔にするツールの作成が望まれる．

参　考　文　献

[1] Stonebraker，M. and Cetintemel，U. :“One Size Fits All : An Idea Whose Time Has Come and Gone”，Proc. of the 21st ICDE，pp．2-11（2005）

[2] Seltzer，M．:“Beyond Relational Databases”，Commun. ACM，Vol. 51，No. 7，pp. 52-58（2008）

[3] 水野正義，宝珍輝尚，野宮浩揮 :“遺伝的プログラミングによる排他制御プログラムの生成について”, 情報処理学会研究報告, 2009-DBS-148（27），2009-FI-95（27）（2009）

[4] 大野友寛，宝珍輝尚，都司達夫 :“排他制御機構の拡張可能化についての一考察”，情報処理学会データベースシステム研究会報告，Vol. 96，No. 11，pp．91-98（1996）