G-15

遺伝的プログラミングを用いた排他制御プログラムの自動生成

Automatic Generation of Concurrency Control Program by Using Genetic Programming

西郷 達也†　　宝珍 輝尚†　　野宮 浩揮†
Tatsuya Saigo　　Teruhisa Hochin　　Hiroki Nomiya

1．はじめに

近年，コンピュータの技術の発展に伴い，扱うデータの種類も多様化している．これらのデータを様々な目的で管理し利用するためには，各用途に対応した機能を持つデータベース管理システム(Database Management System；DBMS)が必要とされる．DBMS はデータ操作を行う際，複数のトランザクション処理を正常に実行できなければならない．この処理を制御する機構として排他制御機構がある．排他制御は悲観的排他制御と楽観的排他制御に大別でき，これらはトランザクションが競合して同じデータに対してアクセスする頻度により使い分けることができる．競合の生じやすい環境で使用される悲観的排他制御法には二相ロック法や時刻印順序法などがある．

異なる分野のデータベースを利用する際，トランザクションにおいても同様に異なる特徴を持つものが発生する．一つのデータを同時に更新することが頻繁にあり得る場合や，反対にデータアクセスの競合の頻度が低い場合などが考えられ，単一の排他制御機構では効率が悪い．それぞれの分野に適した排他制御機構が生成できれば最適な DBMS の構築へと繋がる．

ここで，生物の進化の仕組みに着想を得た遺伝的アルゴリズム(Genetic Algorithm；GA)を拡張した遺伝的プログラミング(Genetic Programming；GP)では，プログラムを生物の遺伝子とみなして進化させることで最適なプログラムを生成することができる．

田村らは，GP を用いて，トランザクションの特徴に応じた排他制御プログラムを生成する試みを行っている [1]．ここでは，任意の排他制御プログラムを生成可能とするために，不定変数と共有不定変数を導入している．不定変数とは，データやトランザクションの情報を保持する変数であり，共有不定変数とはデータとトランザクションの組に関する情報を保持する関数である．初期プログラム木を与える実験と，与えずに排他制御プログラムの生成を行う実験を行い，二相ロック法・ロック法・時刻印順序法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排他制御プログラム木が生成できるという結果が得られている．しかし，初期プログラム木を与えなかった場合，いくつかの評価用スケジュールについては競合等価となるような排他制御プログラムが生成できない結果となっている．また，排他制御として意味のある部分木となるような適切な組み合わせを発生させることを目的として，データが個別に持つ不定変数を操作するノードを使用せず，データとトランザクションがその間で個別に保持する共有不定変数のみを使用する手法を提案しているが，不定変数を使用しないため，任意の排他制御プログラムの生成は不可能となっている．

そこで本論文では，トランザクションの特徴に応じた任意の排他制御プログラムの生成をめざし，不定変数を用いずに任意の排他制御プログラムを構成できるノードの改良を提案する．ここでは，「共有不定変数へのデータの値の格納」を行う方法，「データ自身への共有不定変数の可能化」を行う方法や「システムオブジェクトの導入」を行う方法を提案する．比較検討した結果，「共有不定変数へのデータの値の格納」を行う方法が良いことを明らかにする．そして，この方法に基づいて実装を行い，終端ノードと非終端ノードの合計数をほとんど増加させずに任意の排他制御プログラムを実現できることを示す．

以降，**2.**で GP について概説し，**3.**で先行研究[1]の排他制御プログラム生成システムについて述べる．**4.**では提案手法について述べ，**5.**でノード実装を示す．**6.**で実験を行い，**7.**でまとめる．

2. 遺伝的プログラミング

2.1　遺伝的アルゴリズム

遺伝的アルゴリズム（Genetic Algorithm；GA）は進化論的な考え方に基づいてデータを操作し，自然淘汰のシミュレーションを行い最適化や学習，推論を扱う手法である．GA のアルゴリズムは以下のとおりである．

(1)現在の世代の集団をランダムに生成する．
(2)現在の世代の集団内の各個体に対して適合度を計算する．
(3)適合度を元に現在の世代から個体を選択する．
(4)選択された個体に対して交叉・突然変異などの操作を行い，次の世代の集団を生成する．
(5)(2)～(4)の操作を必要分繰り返し，最終的に現在の世代の集団内で最も適合度の高い個体を解とする．

2.2　遺伝的プログラミング

遺伝的プログラミング（Genetic Programming；GP）は GA における個体の遺伝子型を木構造などの構造的な表現が扱えるように拡張したものである．GP では，突然変異，交叉などの遺伝的操作を部分木の変更によって実現する．また，木構造は主に LISP で用いられるＳ式で表すこともできる．このときの対応関係を図 1 と図 2 に示す．図中の F_1，F_2 は非終端記号を表し，T_1，T_2 は終端記号を表している

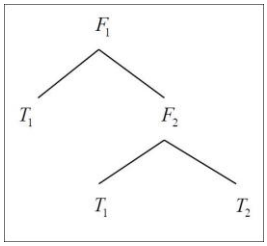

図 1　木構造の例

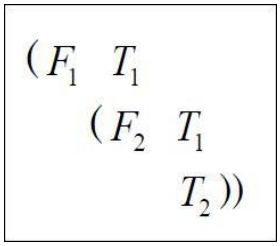

図 2　S 式表現

†京都工芸繊維大学, Kyoto Institute of Technology

3. 排他制御プログラム生成システム

3.1 遺伝的アルゴリズム

本論文で前提とするシステムのシステム構成を図 3 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，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（2）の操作に戻る．最終世代なら終了して最良個体を得る．

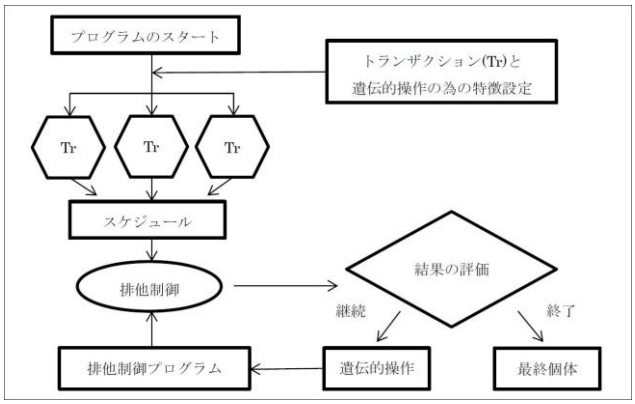

図 3 排他制御プログラム生成システムの概要

3.2 使用ノード

プログラム木を構成するノードには終端記号と非終端記号が存在し，組み合わさって排他制御プログラムとなる．田村らは，個体（プログラム）を構成するノードの基本を変数の操作とし，変数として以下の 2 つを提案している[1].

(1)不定変数

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．データやトランザクションは任意数の不定変数を持つことができる．また，不定変数は自身を示す識別子 ID と格納する値 value を持つ.

(2)共有不定変数

共有不定変数はトランザクションとデータの組毎に一つずつ持つことができる．共有不定変数は，トランザクションの識別子とデータの識別子の組で一意に識別でき，格納する値 value を持つ.

3.3 適合度

まず，直列化可能性を検証する．直列化可能性の検証とは，複数の処理を並行して行った時に，それらの処理を逐次的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，直列化可能性が保証される場合のみ以降の適合度計算を行う．直列化可能性が保証されない場合は適合度を最低に設定する.

適合度の計算に関わるものは大きく以下の 5 項目である.

(1)同時実行性
(2)トランザクションの応答時間
(3)トランザクションのアボート率
(4)プログラムサイズ
(5)全テストスケジュールに対する結果の評価

(1)～(5)までの計算を行い，最終的な適合度 E を求める.

4. 提案手法

4.1 ノード改良の要件

要件を以下に示す.

① 不定変数を使用せずに任意の排他制御プログラムが実現できること
② プログラム木の終端ノード，非終端ノードの数が大幅に増加しないこと
③ 代表的な排他制御プログラムを実現するプログラム木のノード数が大幅に増加しないこと
④ 実装するのがデータベース処理性能に大きな影響を与える排他制御プログラムであるので，実行性能が良いこと
⑤ 実現が可能であること（実現が容易であることが望ましい）

4.2 改良方法

4.2.1 方法 1：共有不定変数へのデータの値の格納

不定変数が持つのと同様な id の変数を持たせることで，データとトランザクションの間に複数の共有不定変数を持つことを可能とし，共有不定変数にデータが持つべき値を格納させる.

・方法 1-1:あるデータに対して保持する値が変更される度に，そのデータとの間に関係を持つ全ての共有不定変数の値を更新する.

・方法 1-2:あるデータに対して，そのデータとの間に関係を持つトランザクションの共有不定変数の値を全て比較し，最大値を持つデータを参照する.

4.2.2 方法 2：データ自身への共有不定変数の可能化

データが自分自身に共有不定変数を持つことを可能にすることで，データの値を不定変数による場合と同様に保持でき，共有不定変数の値を探索する時間も短くなる．しかし，追加するノードが削除したノードと同程度必要となり，追加操作ノードの作成も必要となる.

4.2.3 方法 3：システムオブジェクトの導入

システム全体を表すオブジェクトを導入し，データやトランザクションとの間に共有不定変数を持つことを可能とする．システムとの共有不定変数を得ることで，探索時間は短くなるが，システムに関する操作ノードが新たに必要となる.

・方法 3-1:システムの共有不定変数を操作するノードを新しく作成する.

・方法 3-2:共有不定変数に id を持たせ，その値が負(-1，-2 など)の時の値をシステムとの共有不定変数の値として扱う．この場合，あるデータに対して負の値の id を持つ全ての共有不定変数の値を設定修正する必要がある.

4.3 比較

前述の提案方法を，非終端ノード数，終端ノード数，処理時間・プログラムの修正量の観点から比較する．表 1 に各提案手法での定性的な評価を示す．

表 1 では，「良い」を「○」で示し，「普通」を「△」，「悪い」を「×」で示している．表 1 より，方法 1-2 の手法でデータの値を保持・利用するのが良いと考えられるので，方法 1-2 を採用することとする．

表 1　提案手法の比較

方法	非終端ノードの数	終端ノードの数	処理時間	修正量	総合
1-1	○	○	×	△	△
1-2	○	○	×	○	○
2	×	○	○	△	△
3-1	△	○	○	×	△
3-2	○	△	×	×	△

5. 実装

5.1 共有不定変数の変更

共有不定変数を表す構造体に不定変数の構造体が持つものと同様の id の変数を新しく付け加える．

5.2 ノードの追加と改良

共有不定変数のみで排他制御プログラムを生成するにあたり，不定変数を使用することで可能であった制御プログラムが実現できない場合がある．例えば，時刻印順序法では，オペレーション実行の際に，操作中のデータとトランザクションの間の RTS/WTS の最大値を取得し，実行可能か否かを判断する．このためには，操作中のデータとトランザクション間に存在する共有不定変数の値を比較し，最大値を返すようなノードが必要である．そこで，最大値を得るような操作を行うノードを作成する．

また，共有不定変数に id や値を設定するノード，id をもとに値を得るノードについても，引数を一つ増やし，id の値を引数として取得するように改良する．最終的な使用ノード数を表 2 に示し，追加ノードと変更ノードを表 3 に示す．

表 2:使用ノードの内訳

ノードの種類	先行研究	本研究
共有不定変数操作ノード	12	15
条件判定	11	11
その他ノード	12	12
ADF	6	6
高機能記号ノード	6	0
合計	47	44

表 3 追加･変更ノード

ノード名	引数の数	動作	種別
get_Sh_val_all_State_max	1	現オペレーションが操作するデータが何れかのトランザクションとの間に id= [i]の共有不定変数を持つとき，その各トランザクションとの間の共有不定変の値を比較し，最大の値を得る．	非終端
set_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の id=『引数 1』の値を『引数 2』に変更する．宣言されていなければ id=『引数 1』に対する『引数 2』の値の変数として宣言する．	非終端
get_Sh_val	1	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 id=『引数 1』の共有不定変数の値を得る．宣言されていなければ 0 を得る．	非終端

6. 実験

6.1 初期設定

本実験では先行研究[1]において，正しく動作する排他制御プログラムが生成されなかった場合のトランザクションの設定(設定 1)と，正常動作するプログラムが生成できたトランザクションの設定(設定 2)を用いて比較実験を行った．このトランザクションの設定を表 4 と表 5 に示す．また，この二つの実験で使用した遺伝的プログラミングの設定はどちらも同じものを用いた．これを表 6 に示す．

表 4:トランザクションの設定 1

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 5:トランザクションの設定 2

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	10
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	0.8
p4	トランザクションが発するオペレーションの最大数	10
p5	トランザクションが途中で終了する確率	0.1

表 6:遺伝的プログラミングの設定

設定項目	内容	値
s1	プログラム毎の世代数	1000
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	無し
s4	プログラムの最大深さ	8
s5	プログラムの成長方法	GROW
s6	非終端記号における交叉確率	0.3
s7	終端記号における交叉確率	0.4
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	100
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

表 4 で示すトランザクションの設定は p3 の命令(オペレーション)に WRITE が含まれる確率が 0.4 となっており，競合しやすい条件設定となっている．

6.2 実験結果

トランザクションの設定 1 で実験を行った結果，正常に動作する排他制御プログラムが生成された．プログラムの流れは二相ロック法の制御法と似た構造となっていた．ただし，共有不定変数の有無を判別した後の動作において，共有不定変数が存在した場合の制御が不十分であり，その部分を補うためにもう一度(isREAD)ノードと(isWRITE)ノードを使い，残りの制御を行っている個所が存在した．また，オペレーションが WRITE 時に同じ操作を繰り返し行ってしまう部分木が見られ，プログラムサイズが余計に大きくなっていた．

次に，トランザクションの設定 2 の実験における適合度と先行研究における同実験の適合を表 7 に示す．

表 7:適合度

適合度	先行研究	本研究
平均	5.05	5.05
最大値	6.11	5.88

表 7 より，本提案手法による実験の方が適合度は小さい値となっている．生成されたプログラムの内，最大適合度のプログラムを比較すると，改良ノードで生成されたプログラムのサイズが以前のノードで生成されるプログラムよりも大きくなっていた．この原因として，先行研究で使用されていた高機能記号の存在が考えられる．高機能記号は具体的には「変数の ID，あるいは value を判別し，引数として与えられる ID の value を得る，または更新する」ような意味をもった部分木のノードである． 改良後ノードではこの高機能記号を使用しておらず，共有不定変数の ID に対応した vale を判別し，更新や value を得る度にプログラムが相対的に大きくなったのではないかと思われる．

7. おわりに

本論文では，トランザクションの特徴に応じた任意の排他制御プログラムの生成をめざし，不定変数を用いずに，任意の排他制御プログラムを構成できるノードの改良を行った．共有不定変数へデータの値の格納を行う方法を用いて実装し，使用ノード数を減らすことができた．この改良したノードを用いて実験を行った結果，適合度を高くすることはできなかったが，競合が起こりやすい条件設定での排他制御プログラムを生成することができた．

今後は，遺伝的プログラミングの初期設定を変更した場合の実験を行い，適合度及び生成されたプログラムの構造を調べていく予定である．

参考文献

[1] S.Tamura， T.Hochin， H.Nomiya， ”Concurrency Control Program Generation by Decreasing Nodes of Program Trees in Genetic Programming”， Proc. of 2012 IEEE International Conference on Systems，Man，and Cybernetics(IEEE SMC2012)，pp．1023-1028，2012.

[2] 北川博之， "データベースシステム"， 株式会社昭晃堂， 1996.

[3] 伊庭斉志，"遺伝的プログラミング"，東京電機大学出版，1996.

[4] Sang H. Son and Juhnyoung Lee， "A New Approach to Real-Time Transaction Scheduling，" DTIC Online(オンライン)入手先 <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA465532>（参照 2012-11-10）