社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－ARC－157 (32)
2004－HPC－ 97 (32)
2004／3／3

Jojoによる遺伝的プログラミングの並列化

徳田 拓[†1] 田中康司[†2,†1]
中田秀基[†3,†1] 松岡聡[†1,†4]

遺伝子間相互作用ネットワーク推定問題とは，遺伝子の発現量データ系列から複数の遺伝子間における制御関係を推測するものである．この相互関係は，非線形連立微分方程式によって表現される．これまで，遺伝子間相互作用ネットワーク推定は，S-system 表記の微分方程式を用いたものが一般的であったが，S-system 表記の微分方程式は質量作用則表記の近似式であり，遺伝子間の具体的な相互関係を推定することが困難であった．本稿では，質量作用則に基づいた非線形連立微分方程式表記を採用し，進化的計算の一手法である遺伝的プログラミングを用いて，データ系列から相互作用を示す関数を自動推定するシステムを設計・実装した．このシステムより，与えたデータ系列を再現する微分方程式を得ることができた．また，グリッド環境で Java プログラミングを支援する並列実行環境 Jojo を用いて並列化し，実行時間を短縮することができた．

Parallelization of the Genetic Programming using Jojo

Taku Tokuda,[†1] Kouji Tanaka,[†2,†1] Hidemoto Nakada[†3,†1]
and Satoshi Matsuoka [†1,†4]

Estimating mutual interactions of genetic networks is mainly to infer the mutual control relationships from multiple genes from the gene expression data. Such correlations are typically expressible in the form of nonlinear simultaneous differential equations. However, most work to date has employed S-systems as an expression of such differential equations, allowing only rough approximations of mass actions, and as such it was difficult to determine the actual correlations between the genes. Instead, we formulate the mutual interactions as actual simultaneous partial differential equations, and automatically determine its structure and coefficients using genetic programming (GP) from a given data series. Parallel implementation of the scheme in a Grid environment using our Jojo Grid programming system for Java has resulted in precise determination of the equations in many cases within some reasonable time.

1. はじめに

遺伝子の機能を知る研究の一つとして遺伝子間相互作用ネットワークの推定がある．近年，DNA マイクロアレイや DNA チップなどの細胞内 mRNA 発現量測定技術の発達により，一度に大量の遺伝子発現の有無を調べることが可能になった．遺伝子間相互作用ネットワークの推定とは，実験によって得られた遺伝子の発現量データ系列から，複数の遺伝子間における制御関係の予測するものである．

遺伝子間相互作用推定問題において，扱うデータは遺伝子の発現量データ系列である．ある遺伝子は，何かしらの遺伝子の影響を受け生成されることによりその発現量が増加し，何かしらの遺伝子の影響を受け分解されることでその発現量が減少する．この各遺伝子の相互作用は微分方程式で表現され，n 個の遺伝子の相互作用は，n 変数の n 次連立微分方程式で表現される．この微分方程式を求めるのに人間が直接データに触れ吟味した場合，多くの経験的推測や試行錯誤が必要となり，多大な労力を費やすことになる．そこで，数理モデルの選択，構造・係数・定数などの最適化という形で，計算機を用いて自動化する試みがなされている．微分方程式の表現には，一般質量作用則 (Generalized Mass Action, GMA) に基づく微分方程式表記法が用いられる場合が多い．GMA 方程式は，反応様式が記述に反映されており，直感的な理解がしやすい．また，各パラメータを決定することで，任意の相互関係を示す微分方程式を表現できる．また，現在この分野で最も用いられている S-system(Synergistic System) 方程式[1] は，GMA 方程式のパラメータ数を減らした近似式であり，遺伝的アルゴリズム（Genetic Algorithm, GA）を用いてパラメータの最適化を行う．S-system 方程式では，GMA 方程式に比べ係数の最

†1 東京工業大学 Tokyo Institute of Technology
†2 科学技術振興機構 Japan Science and Technology Agency
†3 産業技術総合研究所 National Institute of Advanced Industrial Science and Technology (AIST)
†4 国立情報学研究所 National Institute of Infomatics

適化が容易であり，各要素の関係度合いを知ることはできるものの，相互作用の表現能力に乏しく，具体的なネットワーク構造を知ることはできない．

そこで我々は，構造の最適化を目的とした遺伝的プログラミング（Genetic Programming, GP）[3] を用いて，相互作用を明確に表現できる GMA 方程式に基づく微分方程式を求める手法を提案する．GP の強力な探索機能により，実際の方程式に近い精密な解を高速に得られることが判明した．また，我々の提案している，グリッド環境で Java プログラミングを支援する並列実行環境 Jojo[2] を用いて，マスターワーカ並列化を行うことにより，16 並列までは容易に並列化を有効に行うことを示せた．今後の実行の改良によりシステム全体の高速化を図れ，それにより高速な推定が今後期待できる．

2. 遺伝的プログラミング

遺伝的プログラミング (以後 GP)[3] は進化的計算の一種である．進化的計算とは生物の集団遺伝・進化の過程を計算機上で模倣して，適応・学習・最適化などの機能を実現しようという手法の総称である．1992 年に John Koza は，プログラムを発展させるために GA の遺伝子型を構造的表現 (グラフ構造や木構造) を扱えるように拡張し，この方法を GP と名付けた．以来，GP は進化的計算の一分野として研究されるようになり，現在では，ロボットプログラミング，人工生命，システム同定問題，人工知能の問題解決・推論・学習，分子生物学など，様々な分野で応用されている[4]．

GP の流れを示す．

(1) 初期母集団を完全ランダムに生成
すべての個体に係数最適化をし，適合度を計算

(2) 終了条件に照らし合わせる
この条件を満たすときは GP を終了
- 終了条件 1: 母集団の中に十分に高い適合度を持った個体が存在する
- 終了条件 2: 規定回数の世代交代を終了

(3) 終了条件を満たさなかった場合
- 子個体を生成
- 係数最適化
- 母集団との入れ替え
- 世代数+1
- (2) に戻る

GP では，親個体の選択，母集団の入れ替えなどに関し様々な方法がある．例として，CCM(Characteristics Collection Model)[5] や，MGG(Minimal Generation Gap) モデル[6] がある．これらのモデルは，元々 GA の世代交代モデルとして提案されたものであるが，GP にも適応可能である．

表 1 ノードと id 番号の対応表

ノード	+	-	×	C	X_0	X_1	...
id	1	2	3	4	5	6	...

3. 遺伝子間相互作用ネットワーク推定への GP の適用

3.1 LinearGP を用いた個体の表現

本システムで GP が対象とする構造は，微分方程式を表現する木構造である．取り扱う問題が n 変数の場合，n 本の微分方程式が必要とされ，n 個の木構造の集合が 1 つの個体となる．

一般的には GP では木構造のプログラムをポインタで表現するが，本システムでは Linear GP[7][8] の表現法を用いる．Linear GP では対象とする遺伝子型を 1 次元配列として扱う．Linear GP の長所は，遺伝的操作を行う際，ポインタでつながれた木構造を操作するのではなく 1 次元配列を操作することになり，これによりポインタ参照のオーバーヘッドを削減し，高速かつ，メモリ消費の小さなシステムが実装可能な点である．

本システムでは，木構造のノードとして非終端記号に演算子 $\{+, -, \times\}$，終端記号に定数・変数 $\{Constant, X_0, X_1, X_2, ...\}$ を用いる．表 1 に示すようにノードの種類別に int 型の id 番号を割り振り，構造を決定する情報として id の配列を用いる．図 1 では，Linear GP における個体の例として，微分方程式とそれに対応する木構造および線形配列表現を示す．微分方程式を表す木構造は 2 分木で表現される．末端のノードには終端記号，節となるノードには非終端記号が割り当てられる．節とその下に付くノードの関係は，演算子の後に引数を並べて書く Prefix 形式で表現する．id 番号を Prefix 形式に従って並べることにより，木構造を配列として表現する．id 配列の中では定数に具体的な値を持たせない．定数部だけの最適化を行うため，id 配列とは別に，id 配列の Constant の数に対応した Constant 配列を用意する．本システムで用いる非終端記号は引数を 2 つ取るものだけなので，全 2 分木になる．

ある配列が，部分木として正しい構造であることを判定するのに，StackCount（Stack に対する push と pop の回数の差．以下 SC）を用いる．非終端記号なら-1，終端記号なら+1 とし，配列の任意の位置から順に足してゆき，その合計が初めて+1 となる位置までの部分列が，部分木として正しい構造を持つことになる．

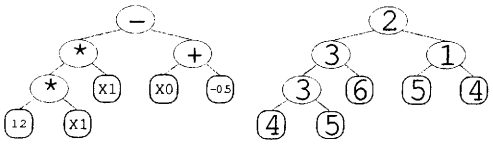
3.2 個体の生成

個体を生成する場合，完全ランダム生成と母集団に基づく生成がある．個体の完全ランダム生成の場合，id をランダムに選択してゆき，SC の合計が 1 になる

微分方程式

$dX_i = 1.2X_0X_1 - X_0 + 0.5$

木構造表現と対応 id



id 配列・Constant 配列・SC 合計

$$\begin{cases} id[\,] = \{2,3,3,4,5,6,1,5,4\} \\ cons[\,] = \{1.2\ ,\ -0.5\} \end{cases}$$

図 1 Linear GP における個体の例

まで，順に id 配列に収めてゆく．問題の次元数によって，木の構造が大きく変わらないように，各 id の出現確率を調整可能にする．$\{+,-,\times,Constant\}$ の出る割合を固定し，残りの割合を各変数が等確率で選択されるようにする．また，木の深さ，配列の長さに制限を持たせ個体の木のサイズを調整可能にする．

子個体生成は，母集団に含まれる個体を基に，その形質を受け継ぐ新個体を作る．まず，母集団から親となる個体をルーレット方式で 1 つ選ぶ．ルーレット方式とは，個体の適合度の高さに応じて選択確率が高まるという選択方法である．この手法により優秀な親の遺伝子を次世代に伝えやすくなる．次に，親個体の各木に対し遺伝的操作を加える．これにより生成される新しい構造を持つものを子個体とする．遺伝的操作には，コピー，交叉，突然変異の 3 種類を用いる．これらは，あらかじめ設定された比に応じてランダムに選択される．

- コピー
  - 親個体の木を，そのまま子個体の木とする
- 交叉
  (1) 交叉させる相手個体を母集団の中からルーレット方式で選び出す
  (2) 親個体の木から部分木を一つランダムに選び，また，相手個体の対応する木から，部分木を 1 つランダムに選ぶ
  (3) 部分木同士を交叉させ，親個体側の木を子個体の木とする
- 突然変異
  (1) 完全ランダム生成で木を一つ作る
  (2) 親個体の木から部分木を一つランダムに選ぶ
  (3) その箇所の部分木と新しく作った木と入れ替えたものを子個体の木とする

3.3 個体の評価と適合度

適合度とは，その個体がどれだけ目的に合っているかを表す指標である．微分方程式の場合，与えたデータ系列にどれだけ近い値が出せるかが問題となる．目的に似ている度合いを適合度とし，適合度を基に個体の評価を行う．本システムでは，与えたデータ系列と得られた微分方程式によるデータ系列の各点での自乗誤差を用いて適合度を導き出す．すべてのデータ点の平均自乗誤差を S とし，適合度 $E1$ を以下の式で定義する．

$$S = \sum_{j=1}^{P}\sum_{i=1}^{N} |F_{exp_{i,j}} - F_{cal_{i,j}}|^2 \ /PN$$

$$E1 = 1/(1 + S \times \mathrm{Weight})$$

ここで，F_{exp} は与えたデータ系列，F_{cal} は得られたデータ系列，P は変数の数，N はデータの数を表す．Weight はあらかじめ定めておくパラメータで，E1 を 0 から 1 の間でバランスよく分散させるため用いる．また，最適化するプログラムの複雑化を防ぐために，複雑性に対しペナルティを与える．具体的には，各木の平均の深さに対してペナルティを課し，深い木構造の評価を低くする．$E1$ に対してペナルティを与えたものを $E2$ とする．

$$E2 = E1 - \mathrm{Depth} \times \mathrm{Penalty}$$

GP の終了条件には $E1$ を評価し，世代交代時の個体評価には $E2$ を評価する．

3.4 係数最適化

ある 1 個体の構造が決定したとき，ある定数配列を持っているが，それらはその構造の可能性を最大限に引き出すものとは限らない．そのため，定数部分を調整して，最大の適合度を返すような定数を決定する必要がある．係数最適化自身 GA，進化プログラミング（Evolutionary Programming, EP），進化戦略（Evolution Strategy, ES）など様々な方法が考えられる．本システムでは，係数最適化に ES を用いた．ES では，取り扱う定数を実数値ベクトルで表し，乱数を加えて局所的なランダム探索を行う．これを個体が保持している Constant 配列に対して行う．

ES の手順は以下のとおりである．

(1) 初期化：個体の中から，定数部分だけを取り出した配列を，親ベクトルとする
(2) 突然変異：親ベクトルに対して乱数を加えた子ベクトルを複数生成する
(3) 評価：各子ベクトルの適合度を求める
(4) 選択：親ベクトル，子ベクトルの中から最大の適合度を出すものを次ループの親ベクトルとする
(5) (2) に戻る

(2) から (4) をループさせることで，その個体の適合度をより高くする定数配列を求めることができる．係数最適化後の適合度を個体の適合度とする．

係数最適化の終了条件は

- 規定回数 A 回のループを完了
- 規定回数 B 回の連続したループにおいて，適合度が上昇しない

の 2 点とする．

親ベクトルに乱数を加える際，以下の条件を満たすように乱数を生成する．

- 適合度が低いときは，近傍に最適解がないと思われるため，広範囲にわたって探索する
- 適合度が高くなると，十分に近傍の探索をする
- 局所解からの脱出も考慮に入れる

微分方程式は4次のRungeKutta法を用いて解く．係数最適化を行う際，候補の定数ベクトルすべて対して微分方程式を解くことになり，これが最も計算量が多くなる部分となる．係数最適化では累積自乗誤差が評価され，これが最小となる個体が最も優秀となる．計算量削減のため，微分方程式をすべて解き終わった後に累積自乗誤差を計算するのではなく，微分方程式を解きながらサンプリングポイントごとに自乗誤差を計算・累積させる．これにより，あらかじめ親ベクトルの適合度を与えておき，累積した自乗誤差がこれを越えた時点で微分方程式の計算を打ち切ることで高速化を図る．

3.5 世代交代と終了条件

世代交代は1子個体生成ごとに行う．子個体と母集団内の個体全ての中で最も低い適合度 $E2$ を持つ個体が淘汰され，残りが次世代の母集団となる．ただし，すべての木に対してコピーのみの操作だった場合は例外として扱う．この場合，親個体と子個体の構造は変化せず，係数最適化だけ行うことになる．先に述べた入れ替え方法だと，母集団内に無駄に同じ構造が増えることになるので，この場合は親個体と子個体の入れ替えとする．また，終了条件は以下の2条件とする．

- 規定値以上の適合度 $E1$ を持つ個体が生成される
- 規定回数の世代交代が終了する

3.6 Jojoによる並列化

本システムでは係数最適化部分を容易に並列化の対象とすることが可能である．係数最適化部分では，微分方程式を定数の候補数だけ解くが，その際には個々の方程式で微小区間の計算を繰り返すので，計算量が比較的多い．また，個々の微分方程式では，初期条件の供与以降は解を得るまで計算を独立して行える．

そこで，我々の提案しているグリッド上の並列プログラミングフレームワークであるJojoを用いて，マスタ・ワーカ方式による並列化を行った．JojoはJavaで実装され，グリッド&クラスタ環境が構成する多階層の実行環境への適用機構，Globusやsshを用いた安全な起動と通信，直感的で並列実行に適したメッセージパッシングAPI，プログラムコードの自動アップロードといった特徴を持つ．マスタ・ワーカ並列化において，マスタ側は個体の生成，母集団の管理，世代交代の評価を担当し，ワーカ側は個体の係数最適化と適合度の計算を担当する．

表2 実験環境

	PrestoIII node	Master node
CPU	AthlonMP 2000+	Intel Mobile Pentium III-M (Low Voltage) 866Mhz
RAM	1024MB	640MB
OS	Linux Version 2.4.22	WindowsXP + Cygwin 2.416

4. 方程式推定の実験および性能評価

実装したシステムの性能を評価するため，単体版と並列版それぞれに対し実験を行った．実験では，まず問題となる微分方程式を設定し，これを解いてデータ系列をとる．次に，このデータ系列から，本システムを用いて微分方程式を推定した．

実験には本研究室のPrestoIIIクラスタのノードを利用した．PrestoIII nodeを一台用いて出力の評価を行い，続いて並列化では，別にマスタ・ノードを用意し，ワーカにPrestoIII nodeを用いた．用いたノードの性能を表2に示す．また，マスタは研究室内の100Base-TのLAN環境に普通に接続され，クラスタの各ノードや専用のファイルサーバなどを接続するサービス側のネットワークに研究室内ルータなどを介して通信している．Jojoでは接続にGlobusやsshを用いる．今回はCygwinを通してsshで接続する．マスタ・ワーカ間のスループットは，上り下り共に約70Mbpsである．

4.1 出力の評価

4.1.1 サンプル問題の方程式

2次の問題として式(1)，3次の問題として式(2)を設定した．

2次の問題

$$\begin{cases} dX_0/dt = 1.2 - X_0X_1 \\ dX_1/dt = 2.2X_0 - X_1 \\ X_0(0) = 4 \quad X_1(0) = 2 \end{cases} \tag{1}$$

3次の問題

$$\begin{cases} dX_0/dt = -1.2X_0 - 0.6X_1 + 0.2 \\ dX_1/dt = 0.5X_0X_1 - 1.6X_1 + 1.2 \\ dX_2/dt = 2.0X_0 + 1.7X_1 - X_2 \\ X_0(0) = 3 \quad X_1(0) = 2 \quad X_2(0) = 1 \end{cases} \tag{2}$$

4.1.2 各種設定

実験ではRungeKutta法の刻み幅を0.5とし，$0 \leq t \leq 10$ の範囲で，0.5ごとに20点で値を測りデータ系列とする．終了条件は，適合度が0.999以上の個体の出現，または5万世代終了時とする．母集団は200個体を保持する．初期母集団は，ランダムに400個体生成したうちの，上位200個体を採用する．遺伝的操作では，コピー・交叉・突然変異が等確率で起こるものとする．適合度の計算におけるパラメータWeightを50，Penaltyを0.03とする．係数最適化では，ループ

表 3　seed 別の終了世代数と実行時間

seed	0	1	2	3	4
終了世代	6593	4503	3942	1916	154
time(sec)	1004	616	638	302	56

表 4　3 次の問題:終了時の適合度分布表

適合度	0.92~	0.94~	0.96~	0.98~ 1
分布	3	14	5	1

ごとに子ベクトルを 30 本生成する．30 ループ終了，または 2 ループの間適合度の上昇が見られない場合をもって係数最適化を終了する．

4.2 GP による方程式推定の結果

4.2.1 2 次の問題

2 次の問題 (式 (1)) を 5 種の乱数 seed で実行した．seed ごとの終了世代数と実行時間を表 3 に示す．5 度の実行すべて場合において，5 万世代内に規定値以上の適合度に達して終了している．

seed4 の出力結果を例として示す．ここでは，定数は小数点以下 2 桁で表示する (切捨て)．

E1 = 0.999382...　E2 = 0.894447...

$$dX_0/dt = (1.19 + < [X_0 \times 0.0] - [X_0 \times X_1] >)$$
$$dX1/dt = < -0.54 - (< X_1 - 0.56 > + ([-3.17 \times X_0] + X_0)) >$$

さらに，この式の括弧を外しまとめると，

$$dX_0/dt = 1.19 + X_0X_1$$
$$dX1/dt = 2.17X_0 + X_1 + 0.02$$

となり，元の式にほぼ一致している．

4.2.2 3 次の問題

3 次の問題 (式 (2)) について，23 種の乱数 seed で実行した．結論として，いずれも規定値以上の適合度には達しなかった．最終的な適合度の分布は表 4 ようになった．

この中で，最優良個体であったものを以下に示す．

E1 = 0.984238...　E2 = 0.885814...

$$dX_0/dt = ([X_0 \times -1.51] + X_1)$$
$$dX_1/dt = [0.14 \times [(X_1 + -0.44) \times (< X_0 - X_2 > + 2.23)]]$$
$$dX_2/dt = ((X_0 + X_1) + (< X_0 - X_2 > + < X_1 - [X_1 \times 0.27] >))$$

これをまとめると，

$$dX_0/dt = -1.51X_0 + X_1$$
$$dX_1/dt = 0.14X_0X_1 - 0.14X_1X_2 - 0.06X_0 + 0.31X_1 + 0.06X_2 - 0.13$$
$$dX_2/dt = 2.00X_0 + 1.73X_1 - X_2$$

となる．

図 2 に与えたデータ系列 (X_{exp_i}) と得られたデータ系列 (X_{cal_i}) を示す．結果の式を見ると，X_2 式の構造は問題式とほぼ同じだが，X_0，X_1 式の構造はかなり異なる．しかし，図 2 を見ると，得られた式のデータ系列は与えたデータ系列にかなり合致している．また，

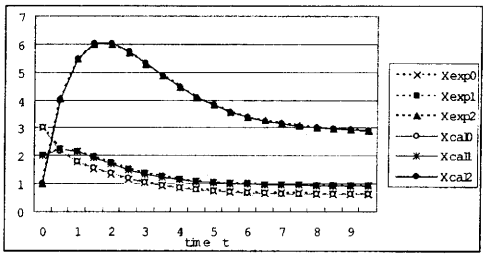


図 2　3 次の問題：与えたデータ系列と得られたデータ系列の比較

表 5　ワーカ上での平均処理時間 (msec)

worker's job	5E-1	1E-1	1E-2
ES time	91	245	1831

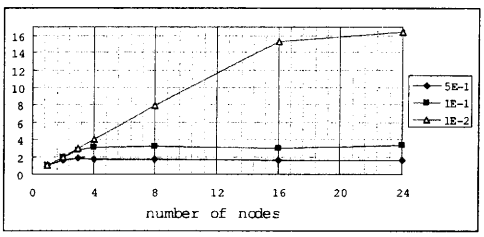


図 3　ワーカ数と並列化効率の関係

適合度は約 1 万世代あたりまでに最終的な適合値に達し，終了条件である 5 万世代までは変化しなかった．

4.3 並列化実行の評価

2 次の問題 (式 (1)) に対して並列化を行い，ワーカで行う RungeKutta 法の刻み幅 を 5E-1，1E-1，1E-2 としてその並列化効率を測定しする．これに先立ち，まず基本的なデータとして，対象となる 2 次の問題の個体をマスタワーカ間で往復させた際にかかる通信時間を測定した．送受信される個体の情報をシリアライズした容量は約 6Kbyte となり，シリアライズ・デシリアライズを含む通信時間は 120msec であった．

表 5 にワーカ上での平均処理時間を示す．刻み幅を小さくすると処理時間が大きくなることがわかる．図 3 の並列化効率では，刻み幅が 5E-1 の場合は約 2 並列，1E-1 の場合は約 3 並列，1E-2 の場合は約 16 並列までスケールしている．これは表 5 のワーカ上の平均処理時間の増加により，刻み幅が小さくなるにつれ，ワーカの処理時間に対して通信時間 (120ms) が相対的に小さくなるので，高い並列化効率が得られたと推測される．

5. 議　　論

2 次の問題において元の id 配列と全く同じものを解として導いたものはなかったが，どの解も式の構造

を整理することで元の式とほぼ同等の構造となった．一方，3次の問題では，得られたデータ系列は与えたデータ系列にほぼ一致するが（図2），式の構造自体が元の式とは異なる解が得られた．逆問題を解く場合，解となる数理モデルは複数あると考えられ，問題式も得られた式もこの複数ある解の一つだと考えられる．問題式を一意に求めたい場合は，より拘束条件を強める必要がある．具体的には，与える問題として一つのデータ系列ではなく，初期値を変化させた複数のデータ系列を用い，これらの差を同時に評価することで，より一意性の高い解が求められると考えられる．

本システムでは親個体をルーレット方式で選び，最も適合度の低い個体を淘汰することで非常に収束性が高くなっている．収束性の高い母集団は局所解に陥りやすく，良くないとされており，今後CCMやMGGモデルで世代交代を行わせた結果と比較する必要がある．

実験で得られる解の式で非常に興味深かった点は，0を掛け合わせることによる部分木のキャンセルである．例えば，構造の一部分に"$(X_i - X_i) \times subtree$"という構造がある場合や，係数最適化により"$0 \times subtree$"という結果になることが多々あった．これにより，$subtree$の構造は全く無視されることになる．つまり，正解と思われる部分構造をもつ全体木は，不必要な部分をキャンセルして正しい構造になる可能性がある．GAなどの数値最適化は扱うデータの一部分が変化した場合，それに応じた数値の変化しか現れない．しかし，GPにおける構造最適化で起こるこの部分木のキャンセルは，それだけで大きな構造の変化だといえ，局所解からの脱出に優れているのではないかと考えられる．

通信にかかる時間のほとんどは，個体のシリアライズ・デシリアライズであると考えられる．通信速度を向上させるため，送受信する情報をできるだけ小さくする必要がある．また，マスタ・ワーカ法での並列化ではマスタに通信が集中するため，マスタの通信時間がボトルネックとなる可能性がある．したがって，マスタ・ワーカ間の通信時間とワーカでの処理時間のバランスを考慮して，ワーカでの処理を設計することが重要である．

6. おわりに

本研究では，GPを用いてデータ系列から相互関係を示す非線形連立微分方程式を求めるシステムを実装し，グリッド環境に対応可能な並列化を行った．2次・3次の例題を解き，データ系列を再現する微分方程式を得た．並列化を行った結果，

今後の課題としては以下が挙げられる．

(1) 多次元の問題を扱う場合，現在のシステムの進化の方法が適さなくなる可能性が考えられる．そのため，CCMやMGGモデルはもちろんのこと，次元数に応じて様々なモデルを実装し，今回の手法と比較・検討しなければならない．

(2) 変数が増えると自由度が増し，一意的な式が得られないことが考えられる．この問題に対しては，複数のデータ系列を与え同時に評価するなど，拘束条件を増やさなければならない．

(3) 本実験では，クラスタ環境で24並列までの実験を行ったが，さらなる大規模並列実行に取り組む．

(4) 今後グリッド環境に適用するにあたり，マスタ・ワーカ間の通信速度がネックになることが考えられる．そのため，同一組織内のノード集団ごとにそれぞれ母集団を保持し，高速な送受信が行えるような階層構造をもったシステムを実現する．

謝辞 本研究は，科学技術振興機構の計算科学技術活用型特定研究開発推進事業 (ACT-JST) 研究開発課題「コモディティグリッド技術によるテラスケール大規模数理最適化」の援助による．

参考文献

1) M.A.Savageau: Biochemical Systems Analysis: A Study of Function and Design in Molecular Biology, Addison-Wesley, 1976.
2) 中田 秀基, 松岡 聡, 関口 智嗣 : Java による階層型グリッド環境 Jojo の設計と実装 , SACSIS 2003 , pp. 113-120, 2003.
3) John Koza : Genetic Programming: On the Programming of Computers by Means of Natural Selection,MIT Press,1992.
4) 伊庭 斉志 : 遺伝的プログラミング, 東京電気大学出版局,1996.
5) Isao Ono, Yuichi Nagata, and Shigenobu Kobayashi : A Genetic Algorithm Taking Account of Characteristics Preservation for Job Shop Scheduling Problems, Proc. of the Intl. Conf. on Intelligent Autonomous Systems (IAS-5), pp. 711-718, 1998.
6) 佐藤浩, 小野功 : 小林重信. 遺伝的アルゴリズムにおける世代交代モデルの提案と評価. 人工知能学会誌, 12, 5, pp.734-744, 1997.
7) Nao Tokui and Hitoshi Iba: Empirical and Statistical Analysis of Genetic Programming with Linear Genome, Proc. 1999 IEEE Intl. Conf. on Systems, Man and Cybernetics (SMC99), IEEE Press, 1999.
8) Erina Sakamoto and Hitoshi Iba: Inferring a System of Differential Equations for a Gene Regulatory Network by using Genetic Programming, Proc. of the 2001 Congress on Evolutionary Computation (CEC2001), pp.720-726, 2001.