遺伝子ネットワーク推定のための並列 GP アルゴリズムの評価

田中康司[†1,†2] 中田秀基[†3,†2]
岡本正宏[†4] 松岡聡[†2,†5]

遺伝子ネットワーク推定とは，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．そこで我々は，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開発した．本稿では，開発したシステムの係数最適化部分に注目し，導入した効率的な探索手法を評価した．その結果，導入した手法は解を効率的に探索できることがわかった．

Evaluation of the Parallel GP Algorithm for Inference of Genetic Network

Kouji Tanaka,[†1,†2] Hidemoto Nakada,[†3,†2] Masahiro Okamoto[†4] and Satoshi Matsuoka[†2,†5]

Inference 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 equation, 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 differential equations, and automatically determine its structure and coefficients using genetic programming (GP) from a given data series. When we evaluated the technique adopted as coefficient optimization of the system, the introduced technique has searched for the equation efficiently.

1. はじめに

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タイムコースデータが得られるようになった．これに伴い，得られたタイムコースデータから，複数の遺伝子間の相互作用を推定する遺伝子ネットワーク推定が，現在バイオインフォマティクスの分野で注目を浴びている．

†1 科学技術振興機構
Japan Science and Technology Agency
†2 東京工業大学
Tokyo Institute of Technology
†3 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
†4 九州大学
Kyushu University
†5 国立情報学研究所
National Institute of Informatics

遺伝子ネットワーク推定は，対象となるすべての遺伝子が正常に発現している野生株や遺伝子破壊実験によりある特定の遺伝子の機能を破壊した破壊株のタイムコースデータから，これを再現する数理モデルを推定することで行う．

遺伝子ネットワークを表す数理モデルとして，べき乗則に基づく S-system モデル[1] がよく用いられており，これまでに S-system モデルを数理モデルに採用し，数値最適化を遺伝的アルゴリズム (Genetic Algorithm, GA) で行う手法が研究・開発されている[2),3)]．しかし，S-system モデルはある物質の生成・分解のプロセスをそれぞれ一つの項で表現する近似式であるため，ネットワーク構造の概略は得られるものの，詳細な構造は分からないという問題が生じる．

そこで我々は，より詳細なネットワーク構造の表現が可能な数理モデルとして，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，微分方程式の構造自体を最適化するためのアルゴリズムとして，遺伝的プログラミング (Genetic Programming, GP) を用い

て，与えた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・開発した[4]．また，本システムはグリッド環境下での利用も想定し，Java プログラミングを支援する並列実行環境 Jojo[5] を用いた並列化も行なった．本稿では，開発したシステムの最適化手法として採用した GP の係数最適化部分に注目し，導入した効率的な探索手法の評価を行なった．

2. 遺伝的プログラミング

遺伝的プログラミング (以後 GP) 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[6]．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[7]．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も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)[8] や MGG (Minimal Generation Gap)[9] モデルがある．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．

3. 並列 GP システム

3.1 個体表現

本システムでは，取り扱う問題が n 変数の場合，n 本の微分方程式の集合を一個体として扱い，GP の個体表現として一般的に用いられているポインタ参照を利用した木構造表現ではなく，一次元配列として表現する Linear GP[10)~12)] を採用した．Linear GP の長所は，遺伝的操作を行う際，ポインタで繋がれた木構造を操作するのではなく一次元配列を操作することにより，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量の小さなシステムが実装可能な点である．

表 1 ノードと id 番号の対応

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

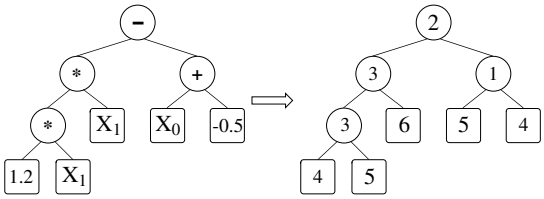

図 1 木構造と id 番号

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 $\{Constant, X_0, X_1, X_2, \ldots\}$ を用い，非終端記号 (演算子) の引数は 2 つとした．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．$dX_i/dt = 1.2X_0X_1 - X_0 + 0.5$ の微分方程式を例として，木構造表現およびノードを id 番号で置き換えたものを図 1 に示す．微分方程式を示す木構造は 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，木構造を一次元配列として表現する．ここで，Constant は定数 (係数) を表し，微分方程式中では double 型の数値として扱うが，id 配列の中では定数であることを示す id 番号 4 という情報のみで具体的な数値を持たせない．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．上記の微分方程式を Linear GP で表現すると id 配列，Constant 配列は，それぞれ id[ ]={2, 3, 3, 4, 5, 6, 1, 5, 4}, cons[ ]={1.2, -0.5} となる．

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 を割り振る．正しい木構造の判定には，非終端記号が引数を 2 つのみとしているため，終端記号の数が必ず非終端記号より 1 つ多くなることを利用する．つまり，配列の任意の位置から順に足していき，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 個体生成と遺伝的操作

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いき，SC の合計が +1 にな

るまで順に id 配列に収めていく．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．$\{+, -, \times, Constant\}$ の出現する割合を固定し，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．各遺伝的操作の詳細な手順は下記の通りである．

- コピー
  親個体の木をそのまま子個体の木とする
- 交叉
  (1) 親個体の木から部分木を 1 つランダムに選ぶ
  (2) 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
  (3) 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
  (4) 部分木同士を交叉させる
  (5) 2 つの木のうち，親個体が基となっている方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 1 つ作る
  (2) 親個体の木から部分木を 1 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 適 合 度

生成されたある個体の微分方程式がどれだけ与えたデータ系列に近いか評価する指標が必要である．これを適合度と呼び，本システムでは下記に示す評価関数から得られる E_1, E_2 の 2 つの適合度を用いた．

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

$$E_1 = 1/(1 + S \times Weight)$$

$$E_2 = E_1 - Depth \times Penalty$$

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, N はそれぞれ変数とデータの数を表す．Weight はあらかじめ定めておくパラメータで，E_1 を 0 から 1 の間でバランスよく分散させるために用いる．また，E_2 は個体の木の平均の深さに対してペナルティを課し，深い木の評価を低くしたものである．GP の終了条件には E_1 を用い，世代交代時の個体評価には E_2 を用いた．

3.4 係数最適化

GP の遺伝的操作によってある個体の構造が決定したとき，この個体の係数部分はその個体において最も適合度が高い係数とは限らない．そのため，係数部分に数値最適化を試み，最適化効率を向上させることにした．これを係数最適化と呼ぶ．係数最適化の手法として，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．本システムでは高速な最適化および実装の容易さから ES を用いた．

ES では，取り扱う係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 の手順は以下の通りである．

(1) 初期化
  最適化対象の個体の Constant 配列を初期親ベクトルとする
(2) 突然変異
  親ベクトルに対して乱数を加えて子ベクトルの集合を生成する
(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，子ベクトルの集合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 Constant 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，

(1) 規定回数 A 回のループを完了
(2) 規定回数 B 回の連続したループにおいて，適合度の上昇が見られない

の 2 つとする．

親ベクトルに乱数を加える際，効率的な探索を行うために以下の条件を満たすことが望ましい．

- 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

本システムでは正規分布に従って乱数を生成するようにし，適合度の値によって標準偏差を変化させた．これにより，生成される乱数の分布が変化し，上記条件を満たすことが可能となる．

さらに (3) において適合度を求める際，候補の定数ベクトルすべてに対して微分方程式の数値計算が必要になる．本システムでは微分方程式の解法に 4 次

の Runge-Kutta 法を用いているが，この数値計算が最も計算コストが高い部分であり，また係数の決定に乱数を用いているため，親ベクトルの適合度よりも高い子ベクトルが得られる可能性も低いと言える．そこで，計算中に子ベクトルの累積自乗誤差が親ベクトルの累積自乗誤差を越えた場合，その時点でその子ベクトルに対しての計算を打ち切り，計算コストの削減を図った．

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すべての中で最も低い適合度 E_2 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では，母集団内に無駄に同じ構造が増えることになるため，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 つとする．

- 規定値以上の適合度 E_1 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

先にも述べたように，本システムでは係数最適化において，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，初期条件を与えれば，係数最適化が終了するまで他の情報の入出力を必要とせず，独立して計算が行える．そのため，マスタ・ワーカ方式の並列化が容易に実装できる．

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なった．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．

4. 実験・考察

4.1 実 験 環 境

実験には東京工業大学学術国際情報センター松岡研究室の PrestoIII クラスタのノードを利用した．用いたノードの性能を表 2 に示す．

マスタ・ノードとワーカ・ノードは同一ネットワークに接続されており，マスタ・ワーカ間のスループットは約 90Mbps，レイテンシは約 0.09msec である．また，係数最適化を除く個体の一往復にかかる時間は約 16msec である (シリアライズ・デシリアライズを含む)．単体版の実験ではワーカ・ノードを一台用いて行い，並列版の実験ではマスタ・ノード一台，ワーカ・ノード複数台用いた．

表 2 実験環境

	Master node	Worker node
CPU	AthlonMP 1900+	AthlonMP 2000+
RAM	768MB	1024MB
NIC	100Base-T	100Base-T
OS	Linux 2.4.22	Linux 2.4.22

4.2 用いた方程式

2 次の例題として式 (1)，3 次の例題として式 (2) を設定した．

2 次の例題

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

3 次の例題

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

実験では，これらの式を Runge-Kutta 法でサンプリングポイント 0.5 ($0 \leq t < 10$) で解いて得られた各変数 20 点のデータをデータ系列として与えた．

4.3 係数最適化の性能評価

4.3.1 適切な標準偏差の値

係数最適化において乱数を生成する際，適合度の値によって標準偏差を変化させるよう設計したが，どのように変化させるのが適切か調べるため，以下のような実験を行なった．

まず，2 次の例題である式 (1) を用いて実験を行なった．この式の Constant 配列は，{1.2, 2.2} の 2 つであり，これらを {0, 0} と設定し，式構造を固定した初期個体を用意し，係数最適化のみを行なった．ここで，ES の 1 ループに生成する子ベクトル数およびループ数は，どちらも 30 とした．実験は，標準偏差を 2.0, 1.0, 0.5, 0.1, 0.05, 0.02 と固定したもの，適合度が 0 の場合の標準偏差を 1.0，適合度が 1 の場合の場合の標準偏差を 0.01 とし，適合度の値によって，この 2 点を結ぶ直線の式から標準偏差を決定するもので行い，それぞれ 10 種の乱数 seed で行なったものの平均を算出した．標準偏差別の適合度の推移を図 2 に示す．

図 2 を見ると，標準偏差が大きいもの，つまり乱数の分散が大きいものは，初期の適合度の成長が早い．しかし，適合度が高くなるにつれ，適合度が上昇しにくくなっている．一方，標準偏差が小さいものほど，初期の成長が遅いものの，全体を通し安定した割合で成長する．これに対し，適合度の値に応じて標準偏差

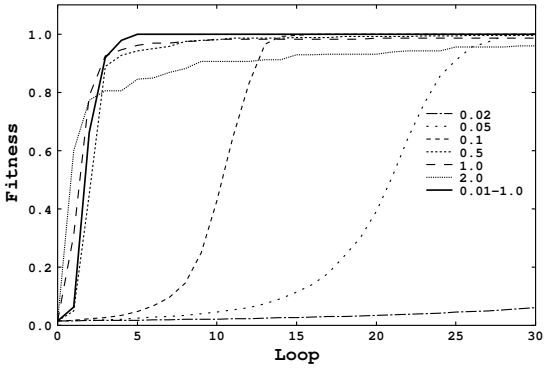

図 2 2 次の例題における標準偏差別の適合度の推移

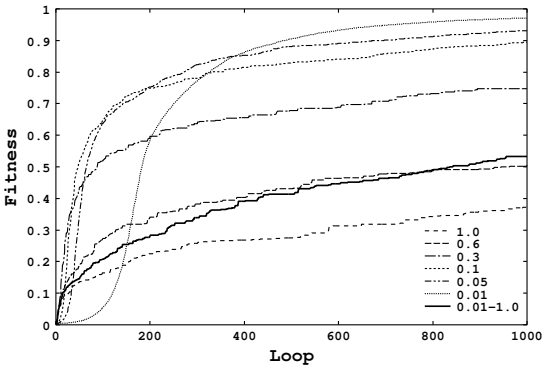

図 3 3 次の例題における標準偏差別の適合度の推移 1

を 1.0 から 0.01 まで動的に変化させたものは，初期の成長も早く，かつ適合度が高い場合でも安定して確実に 1 に近づけることができた．

次に，3 次の例題である式 (2) を用いて 2 次の例題の場合と同様に，すべての Constant 配列を 0 とし，式構造を固定した初期個体を用意して係数最適化のみを行なった．3 次の例題では，ES の 1 ループに生成する子ベクトル数およびループ数は，それぞれ 30, 1000 とした．まず，2 次の例題で最も良い結果を得た適合度の値に応じて標準偏差を 1.0 から 0.01 まで動的に変化させる乱数を用いて実験を行なったところ，1000 ループ終了時点で適合度が約 0.53 となり，高い適合度が得られなかった．そこで，標準偏差を 1.0, 0.6, 0.3, 0.1, 0.05, 0.01 と固定し，適合度の推移を調べてみた．その結果と動的に変化させた場合の推移を図 3 に示す．

図 3 を見ると，標準偏差が 1.0 または 0.6 の場合は，適合度が低いときにはある程度適合度は上昇するが，その成長は全体的に遅い．標準偏差が 0.3 から 0.01 の場合は，標準偏差が大きいほど適合度の低い値からの脱出が速いが，適合度が 1 に近づくにつれ成長が遅くなる．逆に，標準偏差が小さいほど初期の成長は遅いが，適合度が確実に 1 に近づく．以上より，ある適合度の幅ごとに適合度の成長を最も早くする適切な標準偏差があると考えることができる．そこで，各適合度における標準偏差を表 3 のように設定した．また，この新しく設定した標準偏差に基づき乱数を生成した場合の適合度の推移を図 4 に示す．

図 4 を見ると，新しい標準偏差は初期の適合度の成長も早く，また適合度が 1 に近づいても安定した成長をすることがわかる．

4.3.2 最適な子ベクトル数とループ数

先の実験では，ES の 1 ループに生成する子ベクトル数とループ数を 2 次の例題ではどちらも 30，3 次の例題ではそれぞれ 30, 1000 と設定したが，最適な子ベクトル数とループ数を調べるため，3 次の例題に対して，総子ベクトル数を 30000 とし，1 ループに生

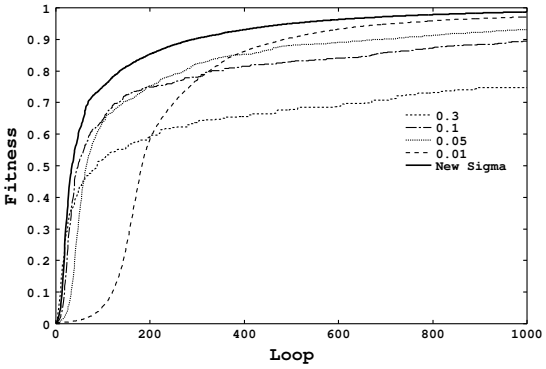

図 4 3 次の例題における標準偏差別の適合度の推移 2

表 4 子ベクトル数・ループ数と最終適合度の関係

子ベクトル数	ループ数	最終適合度
1	30000	0.909001043
5	6000	0.921291509
10	3000	0.975311372
30	1000	0.987062105
60	500	0.980643255
100	300	0.954551963
300	100	0.918596193

成する子ベクトル数とループ数を変化させて最終適合度を調べた．実験は 30 種の乱数 seed で計測し，平均を算出した．結果を表 4 に示す．

表 4 を見ると，1 ループに生成する子ベクトル数が少なすぎても多すぎても最終適合度が上昇せず，最適化効率が悪いことがわかる．これは，ES では子ベクトルは乱数によって生成されるため，1 ループに生成する子ベクトル数が少ない場合は，近傍探索を行うのに十分な適合度を持った個体が生成されにくい．また，子ベクトル数が多い場合はループ数が少なくなるため，十分に近傍探索が行われる前に終了してしまうからであると考える．今回のケースでは，1 ループに生成する子ベクトル数 30，ループ数 1000 が最も最終適合度が高くなった．また，この 30 子ベクトル生

表 3 適合度別の新しい標準偏差

適合度	0 ∼ 0.02	0.02 ∼	0.4 ∼	0.47 ∼	0.9 ∼	0.95 ∼ 1.0
標準偏差	0.3 − 10× fitness	0.1	0.05	0.01	0.005	0.081 − 0.08× fitness

表 5 最適化の各種パラメータ設定

母集団	200 個体
初期 母集団	ランダムに 400 個体生成後, 適合度上位 200 個体を母集団
遺伝的操作	コピー・交叉・突然変異 等確率
最適化 終了条件	適合度 0.999 以上または 5 万世代終了
適合度計算	Weight = 50, Penalty = 0.03
ES の標準偏差	表 3
ES の子ベクトル	ループごとに 30 子ベクトル
ES 終了条件	30 ループ終了または 2 ループ間適合度上昇なし

表 6 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

成・1000 ループにおいて，子ベクトルの累積自乗誤差が親ベクトルの累積自乗誤差を越えた場合に微分方程式の計算を打ち切る手法を導入したところ，微分方程式を最後まで解いたのは全体の約 1.7% であり，導入前に比べ計算量を約 58% に抑えることができ，本手法の導入により，計算コストを削減できることが示せた．

4.4 方程式の推定

4.4.1 最適化の各種パラメータ

最適化の各種パラメータを表 5 のように設定した．

4.4.2 単 体 実 験

2 次の例題

2 次の例題 (式 (1)) に対し 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 6 に示す．5 種の乱数 seed すべてにおいて，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．

$E_1 = 0.999382...$　$E_2 = 0.894447...$

$$dX_0/dt = (1.19 + < [X_0 \times 0.0] - [X_0 \times X_1] >)$$
$$dX_1/dt = < -0.54 - (< X_1 - 0.56 > + ([-3.17 \times X_0] + X_0)) >$$

さらに，この式の括弧を外しまとめると，

$$dX_0/dt = 1.19 + X_0X_1$$
$$dX_1/dt = 2.17X_0 + X_1 + 0.02$$

となり，元の式にほぼ一致している．

3 次の問題

3 次の例題 (式 (2)) に対して 23 種の乱数 seed で実行した．結論として，いずれも 5 万世代内に規定値以上の適合度を得られなかった．最終的な適合度の分布は表 7 のようになった．

表 7 3 次の例題の終了時の適合度分布

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

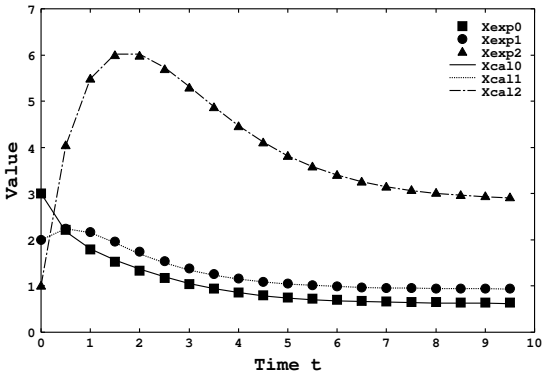

図 5 与えたデータ系列と得られたデータ系列の比較

この中で，最良個体であったものを以下に示す．

$E_1 = 0.984238...$　$E_2 = 0.885814...$

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

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 5 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}) の比較，図 6 に適合度の推移を示す．結果の式を見ると，X_2 式の構造は問題式とほぼ同じであるが，X_0, X_1 式の構造はかなり異なる．しかし，図 5 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 6 を見ると，約 1 万世代までに最終的な適合度に達し，終了条件である 5 万世代まではほとんど変化しなかった．

4.4.3 並列化実験

並列化実験では，2 次の例題 (式 (1)) を用い，ワーカで計算する Runge-Kutta 法の刻み幅を 1/2, 1/4, 1/8 としワーカ側の計算量を変化させ，並列化効率を測定した．これに先立ち，まず基本的なデータとして，対象となる 2 次の例題の個体をマスタ・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

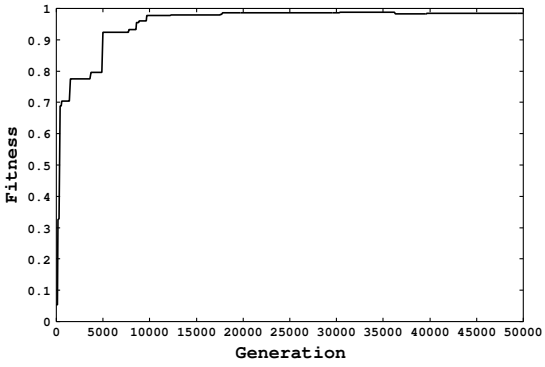

図 6 適合度の推移

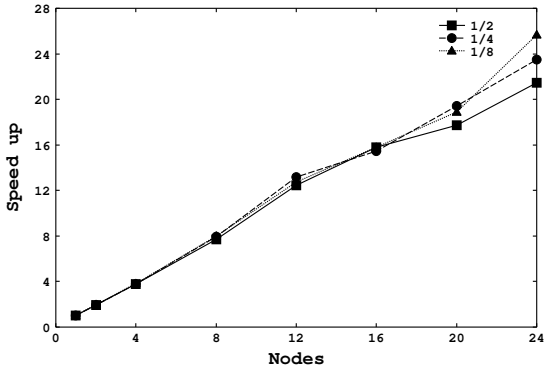

図 8 ワーカ数と並列化効率

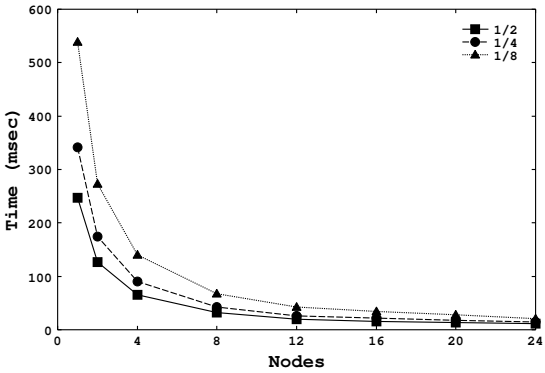

図 7 ワーカ数と ES の 1 個体あたりの平均処理時間 (msec)

れる個体の情報をシリアライズした容量は約 6Kbyte であり，シリアライズ・デシリアライズを含む通信時間は約 16msec であった．

図 7 にワーカ数 (ノード数) と ES の 1 個体あたりの平均処理時間，図 8 にワーカ数 (ノード数) と並列化効率を示す．図 7 を見ると，刻み幅を小さくするにつれ処理時間が長くなり，ワーカ数 (ノード数) に応じて計算時間が減少することがわかる．また，図 8 を見ると，刻み幅が 1/2 では 16 並列，刻み幅が 1/4, 1/8 では 24 並列までスケールしていることがわかり，高い並列化効率が得られたと言える．

4.5 議　　論

係数最適化において適切な標準偏差の値を調べる実験では，適合度が低い場合，広域探索が可能な標準偏差が大きい乱数が望ましく，適合度が高くなるにつれ，近傍探索が可能な標準偏差が小さい乱数が望ましいことがわかり，これらを組み合わせ動的に変化させることで効率的な探索ができることがわかった．2 次の例題では Constant 配列の要素は 2 つ，3 次の例題では 8 つであった．この 2 つの例題を比べると，2 次の場合はある程度大きな標準偏差でも高い適合度まで成長し，かつ成長も早いが，3 次の場合は適合度が低いうちから小さな標準偏差が必要とされ，成長も非常に遅かった．これは，3 次の例題では 2 次の例題に比べ，多くの Constant 配列を最適化しなければならず複雑さが増したことが原因であると言え，問題の複雑さが増した場合，標準偏差の小さな乱数から大きな乱数への変化を緩やかにしていくことが必要であると考えられる．

また，1 ループに生成する子ベクトル数とループ数の関係を調べる実験では，1 ループに生成する子ベクトル数を少なく，ループ回数を多くした方が，標準偏差を動的に変化させる手法がうまく機能し，効率的な探索が可能になることがわかった．しかし，1 ループに生成する子ベクトル数が少なすぎると，初期段階に局所解に陥りやすく，大域探索がうまく機能せず，逆に多すぎると近傍探索がうまく機能しない．問題の複雑さによって適切な値を設定しなければならないと言える．

方程式の推定実験では，2 次の例題において元の 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 次の例題では，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 5)，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 1 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 1 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．

並列化実験では，今回は例題が単純なため，微分方

程式解法に用いている Runge-Kutta 法の刻み幅を小さくすることで，ワーカ側での計算量を増やした．24 並列までの実験では，ワーカ数 (ノード数) に応じて計算時間が減少し，高いスケーラビリティを示すことができた．

5. おわりに

本稿では，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自動導出するシステムの最適化手法として採用した GP の係数最適化部分に注目し，導入した効率的な探索手法の評価を行なった．係数最適化において乱数を生成する際，適合度の値によって標準偏差を動的に変化させたところ，問題の複雑さが増すごとに小さな標準偏差から大きな標準偏差へ緩やかに変化させる必要があることがわかった．また，1 ループに生成する子ベクトル数とループ数も同様に問題の複雑さによって適切な値を設定する必要があることがわかった．さらに，この導入した手法によって，効率的な探索が行なえることがわかった．方程式の推定では，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．また，並列化を行なった結果，高いスケーラビリティを示すことができた．今後の課題としては以下が挙げられる．

- 今回，係数最適化には ES を用いたが，今後大規模な系に適用するにあたり，最適化対象の複雑さが増すことになり，今回導入した標準偏差の動的変化や適切な子ベクトル数とループ数の設定では，効率的な探索が難しくなることも考えられる．そのため，係数最適化に他の最適化手法を用いたものと比較を行い，係数最適化の性能を上げていかなければならない．
- 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，拘束条件を増やす必要があり，複数のデータ系列を同時に評価できるよう，システムを改良しなければならない．
- 本実験では，クラスタ環境で 24 並列までの実験を行なったが，さらなる大規模並列実行に取り組む．また，グリッド環境に適用するにあたり，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，高速な送受信が行なえるような階層構造を持ったシステムを実現する．
- 今回，微分方程式の解法に 4 次の Runge-Kutta 法を用いたが，遺伝子ネットワーク推定への適用を考えた場合，スティフな微分方程式となるため，Runge-Kutta 法では解くことが困難であると予想される．そのため，スティフな微分方程式を解くことが可能な解法の導入の検討が必要である．

これらの課題をクリアすることで，大規模な系の方程式の自動導出が可能となり，さらには遺伝子ネットワークの推定に適用できるようになると期待される．

謝辞 本研究は，科学技術振興機構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) Savageau, M.A.: *Biochemical Systems Analysis: A Study of Function and Design in Molecular Biology*, Addison-Wesley (1976).
2) Ueda, T., Ono, I. and Okamoto, M.: Development of System Identification Technique Based on Real-Coded Genetic Algorithm, Genome Informatics, Vol.13, pp. 386-387 (2002).
3) 上田尚学, 小野功, 岡本正宏: 実数値 GA に基づく効率的な数値最適化手法の開発: 逆問題における多変数最適化への応用, 第 9 回 MPS シンポジウム, pp. 127-134 (2003).
4) 徳田拓, 田中康司, 中田秀基, 松岡聡: Jojo による遺伝的プログラミングの並列化, 情報処理学会研究報告, 2004-ARC-157 2004-HPC-97, pp. 187-192 (2004).
5) 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, SACSIS2003, pp. 113-120 (2003).
6) Koza, J.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press (1992).
7) 伊庭斉志: 遺伝的プログラミング, 東京電機大学出版局 (1996).
8) Ono, I., Nagata, Y. and Kobayashi, S.: 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).
9) 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, Vol. 12, No. 5, pp. 734-744 (1997).
10) Keith, M.: *Genetic Programming in C++: Implementation Issues*, Advances in Genetic Programming, MIT Press, pp. 285-310 (1994).
11) Tokui, N. and Iba, H.: Empirical and Statistical Analysis of Genetic Programming with Linear Genome, Proc. 1999 IEEE Intl. Conf. on Systems, Man and Cybernetics (SMC'99), IEEE Press (1999).
12) Sakamoto, E. and Iba, H.: 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).