遺伝的アルゴリズムを用いた並列グラフ分割アルゴリズム

丸山 勉† 小長谷明彦† 小西弘一†

本論文では，遺伝的アルゴリズムを用いた新たな並列グラフ分割アルゴリズムを提案し，このアルゴリズムを用いることによって，従来のヒューリスティックアルゴリズムと比べてより良い解をより高速に求めることができることを示す．この並列グラフ分割アルゴリズムは，従来のヒューリスティックアルゴリズムと遺伝的アルゴリズムを組み合わせたものであるが，より高速な処理を実現するために，(1)非同期型細粒度並列遺伝的アルゴリズムおよび(2)新たなグラフ分割問題向き交叉オペレータを用いている．15プロセッサによる並列処理では 14〜18 倍の性能向上を確認した．また，本論文で提案する細粒度並列遺伝的アルゴリズムと，複数のコロニーを用いる粗粒度並列遺伝的アルゴリズムを組み合わせることによって，より高並列な処理が実現できることをシミュレーションにより示す．

A Parallel Graph Partitioning Algorithm Based on Genetic Algorithms

Tsutomu Maruyama,† Akihiko Konagaya† and Kouichi Konishi†

We propose a new parallel algorithm for graph partitioning which combines a genetic algorithm and the mincut algorithm. In this algorithm, all genetic operators are designed to be asynchronous in order to exploit maximum parallelism, and genetic recombination operators are specially designed for the mincut algorithm. These genetic operators can substantially reduce the number of cut nets within a reasonable amount of time. The speedup by this algorithm is about 14〜18 times using 15 processors. The scalability of this algorithm is limited by the speed of the broadcasting. In order to exploit more parallelism, we can combine the algorithm with the multi-population strategies. Then, the algorithm achieves high performance in each population.

1. はじめに

本論文では，遺伝的アルゴリズムを用いた新たな並列グラフ分割アルゴリズムを提案し，このアルゴリズムを用いることによって，従来のヒューリスティックアルゴリズムと比べてより良い解をより高速に求めることができることを示す．

この並列グラフ分割アルゴリズムは，従来のヒューリスティックアルゴリズムと遺伝的アルゴリズム(GA)を組み合わせたものであるが，より高速な処理を実現するために，非同期型細粒度並列 GA を提案し，同期型の並列GAと比べて約2倍の速度向上を実現している．また，より良い解が高速に得られるように，新たなグラフ分割問題向き交叉オペレータを用いている．

この並列グラフ分割アルゴリズムは共有メモリ型の並列マシンである Symmetry 上に実装されており，15 プロセッサを用いて 14〜18 倍程度の性能向上率を確認している．また，本細粒度並列GAと複数個のコロニーを用いる粗粒度並列GAを組み合わせることによって，より高並列な処理を実現できることを示す．

なお，本研究は第5世代コンピュータプロジェクトにおいて並列オブジェクト指向言語 A'Um[15] の評価の一環として行われたものである．A'Um ではプロセス間の同期制御等が不要であるためプロトタイピングを容易に行うことができるという利点がある．

2. グラフ分割問題

グラフ分割問題とは，幾つかのノードとノード間を結ぶ何本かのリンクで構成されたグラフのノードを幾つかのグループに分割し（各グループに割り当てられるべきノード数（重み）は決められている），グループ間に跨るリンク数が最小となるような分割を求める問題である．グラフ分割問題は LSI 設計等において非常に重要な問題であるが，この問題の全解探索の計算

† 日本電気(株) C&C システム研究所
C&C Systems Research Laboratories, NEC Corporation

量は膨大であり，実用的な規模の回路において最適解を求めることは困難である．

グラフ分割問題においては，ヒューリスティックアルゴリズムとして文献 1, 2 等の Mincut アルゴリズムと呼ばれるものが有名であるが，これらのアルゴリズムはしばしばローカルミニマニに陥り十分によい解を得ることができない．このため，これらのアルゴリズムを改良したものが数多く提案されている[3)~6)]．これらのアルゴリズムは基本的にグラフを2分割するためのものであり，これを繰り返し適用することによって，任意の数のグループへの分割が実現されている．

以下に，Mincut アルゴリズム[2)]の概略を述べる（以下 MC と呼ぶ）．K 個のノード N_i $(i=1, K)$ からなるグラフを2つのグループ G_0, G_1 に2分割するものとする．ただし，各ノードは重さ w_i $(i=1, K)$ を持ち，$W(G_0)$, $W(G_1)$ はそれぞれ G_0, G_1 の中のノードの重さの総和を表すものとする．

1. 乱数等を用いて各ノードを G_0, G_1 のどちらかに割り振る．
2. 各グループ内のノード N_i を $GAIN(N_i)$ によってソートする．ただし
   $GAIN(N_i)=$
   （Ni から他グループ中のノードへのリンク数の総和$-Ni$ から同グループ中のノードへのリンク数の総和）
3. $W(G_0)>W(G_1)$ ならば，G_0 から G_1 へ，そうでなければ G_1 から G_0 へ，GAIN の最も大きなノードを移動する．
4. 移動されたノードとの間にリンクを持つノードに関して GAIN を再計算し，ソートし直す．既に移動されたノードは，ソートおよび移動の対象から外す．
5. 3および4の処理をすべてのノードが移動されるまで繰り返す．この間で最もグループ間のリンク数が少なくなった状態を保持しておく．
6. 保持した状態を初期状態として，2から5の処理を繰り返す．初期状態より，より良い状態が見つからなくなったら処理を終了する．

MC においては，このようにノードを1つずつグループ間で移動するため，連続的な変化では到達できない解，すなわち複数個のノードを同時に移動しなければならないような解に到達することが難しいという問題点がある．このため初期分割を変えながら何回か試みるという手法がとられることが多い．

これに対し IICME[3)] はノードをクラスタリングすることによって，複数個のノードを同時にグループ間で移動することを狙ったものである．以下 HCME の概要について述べる．以下の記述では，説明を簡単にするためにノード数を $K=2^n$ とする．

1. 互いの間のリンク数が多いノードを2つずつひとまとめにして，
   $C_{1,i}$ $(i=1, K/2)$
   とする．
2. 次に，$C_{1,i}$ $(i=1, K/2)$ を対象として，互いの間のリンク数が多いもの同士をひとまとめにし，$C_{2,i}$ $(i=1, K/4)$ とする．この処理を繰り返すことによって，
   $C_{1,i}$ $(i=1, K/2)$,
   $C_{2,i}$ $(i=1, K/4)$,
   …
   $C_{n-2,i}$ $(i=1, 4)$,
   $C_{n-1,i}$ $(i=1, 2)$,
   ができる．
3. $C_{n-1,1}$ 中のノードが G_0, $C_{n-1,2}$ 中のノードが G_1 に所属するものとする．
4. $C_{n-1,i}$ $(i=1, 2)$ を1段分解して，$C_{n-2,i}$ $(i=1, 4)$ を取り出す．$C_{n-2,i}$ は，$C_{n-1,i}$ に対応してどちらかのグループに所属している．
5. $C_{n-2,i}$ を対象として MC を適用する．
6. 4, 5 の1段分解，MC の適用を $C_{1,i}$ $(i=1, K/2)$ に対して MC が適用されるまで繰り返す．
7. 最後に，$C_{1,i}$ をノードのレベルに分解し，MC を適用する．

このように複数個のノードをクラスタリングし同時に移動することによって，MC より，より良い解を得ることができる．HCME は基本的には決定的なアルゴリズムではあるが，クラスタリングのペアの選択において同一のリンク数を持つものが一般に複数存在するため，どれを選ぶかによって結果が異なる．このため，ペアの選び方を変えながら何回か試行することによって，より良い解を得ることができる．

3. 遺伝的アルゴリズム

以下，遺伝的アルゴリズム（Genetic Algorithms, GA と略す）の概略について述べる．

3.1 遺伝的アルゴリズム概略

遺伝的アルゴリズム（GA）は，複数個の値（個体と呼ばれる）を用いて，これらの個体の間で以下の処理

を繰り返すことによって，より良い解の探索を行う確率的探索アルゴリズムである[7]．この1回分の処理を世代と呼ぶ．

- 交叉 (Crossover または Recombination)
- 突然変異 (Mutation)
- 選択 (Selection)

交叉は，2つの個体の間でその一部を交換し，新たに2つの個体を生成する操作である．交叉は，より良い部分解同士を組み合わせることによって，さらに良い個体が生成されることを期待するものである．突然変異は，個体の一部を確率的に変化させるものであり，探索空間上において，ある個体の近傍の探索を狙ったものである．交叉，突然変異は確率的に行われ，その確率は問題の種類に応じて調整される．選択は，複数個の個体の中でより良い評価値を持つものを選択的に複製し，より悪い評価値を持つものを消去する操作である．選択も確率的に行われ，比較的悪い評価値を持つ個体も，その評価値に応じて低い確率ではあるが，ある程度の確率で生き残る．GA では，このように比較的悪い評価値を持つ個体もある程度残しながら次第に探索範囲を狭めていくことによってローカルミニマムに陥ることを防いでいる．

GA は，一般にヒューリスティックアルゴリズムと組み合わせて用いると，より良い性能を発揮することができる．GA は確率的に各個体を変化させながら探索を行うため，ある程度の値には比較的容易に近付くが（グローバルな山登り），ローカルな山登りは比較的苦手である．そこで，GA とヒューリスティックアルゴリズムを組み合わせ，ローカルな山登りをヒューリスティックアルゴリズムによって行い，グローバルな山登りにGAを利用することによって，より効率的な探索を行うことができる[9),12)]．

GA の交叉，突然変異，選択の個々の処理に関しては，それに非常に近いものを従来のヒューリスティックアルゴリズムにみることができる（例えばグラフ分割問題では，文献1において交叉にかなり近い手法が提案されている）．また，選択処理も基本的には探索木の刈り込みに相当するものであり，よく知られた手法の一種である．GA は生存競争のシミュレーションという観点から発生した探索アルゴリズムであるが，結果として従来からの手法に確率的要素を付け加えて，それらを統一した確率的探索アルゴリズムとなっている．

3.2 並列遺伝的アルゴリズム

GA が複数個の個体を用いて探索を行うため，探索時間がかなり大きくなるという問題点があるが，各個体の独立性が比較的高いため並列処理に適したアルゴリズムであり，幾つかの並列アルゴリズムが提案されている[8)~14)]．

これらは，粗粒度並列処理と細粒度並列処理の2つに分類することができる．粗粒度のものは，一般に全体をプロセッサ個数分のコロニーに分割し，コロニー間で適当に個体を交換し合うことによって探索を行う．すなわち，1つのプロセッサで複数個の個体を扱い，交叉，選択はプロセッサ内で行う．文献 10, 14 では，コロニー間で適度に個体の交換を行うことによって，全個体を1つのコロニーとして扱った場合と同程度の性能が得られることが報告されている．

細粒度のものは，基本的には1個体を1プロセッサが担当し，交叉，選択等の処理はすべてプロセッサ間で行われる．細粒度並列の場合には，交叉，選択は通信のオーバヘッドおよび各処理の待ち合わせ時間等を軽減するために近傍のプロセッサ間のみで行われることが多い．しかし，近傍間でのみ交叉，選択を行うと，交叉，選択の対象となる個体数は，一般に数個程度と非常に少ないため，探索の途中にある程度良い個体が出現するとその影響が非常に大きく現れ，局所解に陥る危険性が高くなるという問題点がある．

また，細粒度並列処理では各処理の待ち合わせ時間を低減するために，選択を非同期化した方式[9)]が提案されている．文献 9 では選択の処理が大幅に変更されており，近傍の中で最も悪い値を持つ子孫が生成された場合には親をそのまま保持し，より良い子孫が生成されるまで交叉，突然変異を繰り返すという方法をとっている．しかし，この方法では探索の途中である程度良い個体が生成されると，その個体がそのプロセッサ中にかなり長い間留まるため，その個体が何回も交叉されることによって，その影響が他の個体に大きく表れ，局所解に陥りやすくなるという問題点がある．

4. 並列グラフ分割アルゴリズム

本並列アルゴリズムは，ヒューリスティックアルゴリズム (MC) と遺伝的アルゴリズム (GA) とを組み合わせた並列アルゴリズムであり，従来のGAとは以下の2点が異なる．

- 非同期型細粒度並列 GA

- グラフ分割問題向きオペレータの導入

GA の非同期化においては，各プロセッサ中の個体の複製を保持するためのバッファを設け，交叉・選択処理にこのバッファ中の値を用いることによって，遺伝的オペレータにおけるすべての同期を取り除いている．また各個体の評価値等をプロセッサ間でブロードキャストすることによって，選択処理の対象となる個体群中の個体数をできるだけ大きくし，局所解に陥る危険性を低減している．また，グラフ分割問題向きオペレータは，交叉および突然変異によって，適度な量のノードがグループ間で交換され，Mincut アルゴリズムの問題点を補うように設計されている．

以下，本アルゴリズムについて詳しく述べる．

4.1 非同期型細粒度並列 GA

以下，非同期型細粒度並列GAの詳細，およびスケラビリティについて述べる．

4.1.1 アルゴリズム詳細

GA において細粒度の並列化を行った場合に並列化による速度向上を阻害する要因は，交叉による2個体間の待ち合わせ，選択におけるN個の個体間の待ち合わせの2点である．GA をヒューリスティックアルゴリズムと組み合わせた場合，ヒューリスティックアルゴリズムの適用に要する時間が各個体ごとにかなり異なるため上記における待ち時間が大きな問題となる．

本論文で提案する並列アルゴリズムでは，以下のようにしてGAにおける同期をすべて取り除いている．

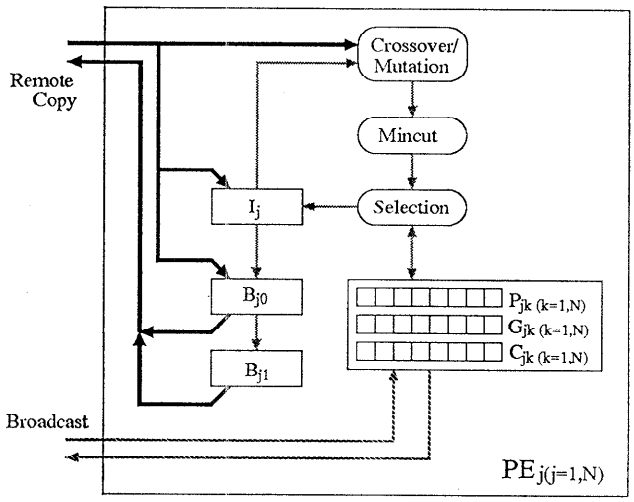
1. 各プロセッサ中の個体はプロセッサ内で各世代ごとに複製される．各プロセッサ中の個体に対して交叉，突然変異，MC 等が適用されている間は，この複製が他のプロセッサから参照される．
2. 交叉は各プロセッサ中の個体と他のプロセッサ中の複製との間で行う．このとき，従来のGAのように2つの個体から2つの子孫を生成するのではなく，1つの子孫のみを生成する．
3. 選択はあるプロセッサ中の個体と他のプロセッサ中の複製との比較という形で行われる．各複製の評価値はプロセッサ間でブロードキャストされているため，選択を各プロセッサ内でローカルに行うことができる．各プロセッサ中の個体が生き残れなかった場合には，他のプロセッサ中の複製の中から比較的良いものが選ばれ，コピーされる．

以下アルゴリズムの詳細について述べる．各プロセッサ PE_j $(j=1, N)$ は，以下のデータを保持している（**図 1**）．

I_j : 個体
B_{j0}, B_{j1} : I_j の複製
P_{jk} $(k=1, N)$: PE_k 中の個体 B_{k0} の評価値
G_{jk} $(k=1, N)$: PE_k 中の個体 B_{k0} の世代番号
C_{jk} $(k=1, N)$: PE_k 中の個体 B_{k0} が他のプロセッサによってコピーされた回数

世代番号とは，そのプロセッサにおいて何番目に生成された個体であるかを表す番号である．B_{k0} $(k=1, N)$ の評価値，世代番号およびコピー回数がプロセッサ間でブロードキャストされ，それぞれ P_{jk}, G_{jk}, C_{jk} $(k=1, N)$ に格納される．各データ構造は次のように初期化される．

I_j : 乱数を用いたグラフの2分割に MC を適用したもの
B_{j0} : I_j の複製
B_{j1} : Empty



I_j : Individuals.
B_{j0}, B_{j1} : Copy of the Individuals.
$P_{jk\ (k=1,N)}$: Performance of $B_{k0\ (k=1,N)}$.
$G_{jk\ (k=1,N)}$: Generation of $B_{k0\ (k=1,N)}$.
$C_{jk\ (k=1,N)}$: Copy Count of $B_{k0\ (k=1,N)}$.

図 1 非同期型細粒度並列 GA 概略
Fig. 1 Overview of asynchronous genetic algorithm.

P_{jk} $(k=1, N)$: B_{k0} $(k=1, N)$ の評価値
G_{jk} $(k=1, N)$: 0
C_{jk} $(k=1, N)$: 0

各プロセッサ PE_j $(j=1, N)$ の処理は，以下のサイクル（世代）を繰り返すことによって進められる．

1. 世代番号＝世代番号＋1
2. (0〜1) の範囲で乱数を1つ生成し，その値があらかじめ指定された交叉確率未満であれば，個体 I_j に対して交叉オペレータを適用し，それ以上であれば突然変異オペレータを適用する．
   - 交叉オペレータを適用する場合
     (a) 交叉の相手 B_{k0} $(k=1, N, k\neq j)$ を1つランダムに選ぶ．
     (b) プロセッサ PE_k から B_{k0} を読み出し，個体 I_j の一部を置き換える．
   - 突然変異オペレータを適用する場合
     個体 I_j に対して突然変異オペレータを適用する．
3. 個体 I_j に MC を適用する．
4. P_{jk}, G_{jk}, C_{jk} $(k=1, N)$ をそれぞれ $Ptmp_{jk}$, $Gtmp_{jk}$, $Ctmp_{jk}$ $(k=1, N)$ にコピーする．この間 P_{jk}, G_{jk}, C_{jk} の更新は禁止される．
5. 個体 I_j の評価値を $Ptmp_{jj}$ に代入する．
6. $Ptmp_{jk}$ $(k=1, N)$ を用いて，I_j および B_{k0} $(k=1, N, k\neq j)$ の適応度を計算する．例えば個体 I_j の適応度 RF_j は，

$$RF_j=\frac{W_j-Ptmp_{jj}}{W_j-Av_j}$$

W_j：過去 M 世代の間の P_{jk} $(k=1, N)$ の最悪値

$$Av_j:\left(\sum_{l=1}^{N} Ptmp_{jl}\right)/N$$

で計算される．この計算式は，評価値が W_j である個体の生存確率が0，評価値がちょうど平均値である個体の生存確率が1となるように正規化を行うものである．評価では $M=5$ を用いた．

7. (0〜1) の範囲で乱数を1つ生成し，その値が RF_j 以下であれば個体 I_j は選択されたことになり，そのまま次の世代へと渡される．RF_j より大きかった場合には，他のプロセッサ PE_k $(k=1, N, k\neq j)$ から B_{k0} がコピーされる．
   - 選択された場合
     B_{j0} を B_{j1} にコピーし，次いで I_j を B_{j0} にコピーする．また，B_{j0} の評価値および世代番号をすべてのプロセッサ PE_l $(l=1, N)$（自分自身も含む）にブロードキャストする．PE_l 中の P_{lj} および G_{lj} はブロードキャストされた値に更新され，C_{lj} は0にリセットされる．
   - 選択されなかった場合
     (a) B_{k0} $(k=1, N, k\neq j)$ を1つを選ぶ．より良いものがより高い確率で選択されるように，B_{k0} は以下のようにして選択される．
     i．(0〜$RFsum_j(N)$) の範囲の乱数 R を1つ生成する．ただし，

$$RFsum_j(m)=\sum_{l=1, l\neq j}^{m} max(RF_l-Ctmp_{jl}, 0)$$

とする．$Ctmp_{jl}$ は，B_{l0} がコピーされた回数であり（選択が開始された時点での値），より良い評価値を持つ個体が何回も繰り返しコピーされることを防ぐために用いられている．GA における適応度とは，ある個体の複製が次の世代において幾つ存在すべきかを表す数である．このため，コピーされた回数を適応度から引くことによって，適応度に相当する分だけ，その個体がコピーされることが期待できる．

     ii．$RFsum_j(k-1)<R<RFsum_j(k)$ となる k を選ぶ．
     (b) B_{j0} を B_{j1} にコピーする．
     (c) $Gtmp_{jk}$ と G_{jk} を比較する．一致した場合には，B_{k0} を I_j, B_{j0} にコピーする．また，値 $C_{jk}+1$ をすべてのプロセッサ PE_l $(l=1, N)$ にブロードキャストし，C_{lk} を更新する．この世代番号の比較からブロードキャストによる C_{lk} の更新までの間は，他のプロセッサからのブロードキャストによる C_{lk} の更新が行われてはならない．本論文の評価では共有メモリマシン上に本アルゴリズムを実装したので，ロック機構を用いて排他制御を行った．遅延時間の大きなネットワークの場合には，世代番号も同時にブロードキャストし，ブロードキャストされた側でも世代番号の比較を行い，世代番号が等しいかそれ以上の場合のみ更新を行う等の方法がある．

     一致しない場合には，B_{k1} が I_j, B_{j0} にコ

ピーされる．ブロードキャストは行われない．一致しない場合とは，PE_j における選択処理中に，PE_k 中の B_{k0} が更新された場合である．この場合には，PE_j 中の評価値 $Ptmp_{jk}$ に対応する個体 B_{k1} がコピーされなくてはならない．また，既に新しい B_{k0} に関するブロードキャストが PE_k から行われているので，PE_j は古い B_{k0} の情報に基づくブロードキャストを行ってはならない．

（d） B_{j0} の評価値，世代番号，十分に大きな値 Z の3つをすべてのプロセッサにブロードキャストする．各プロセッサ PE_j ($l=1, N$) 中の P_{lj} には B_{j0} の評価値が，G_{lj} には世代番号が，C_{lj} には Z が代入される．B_{j0} は，既に他の個体のコピーであるため，それ以上コピーされないように十分大きな値 Z（実際には数十で十分である）が用いられる．

8. 1に戻る．

本アルゴリズムは，共有メモリ型のマシンに実装されているため，ブロードキャストの速度がMCの適用時間等と比べて十分に速い．このため，古い I_j を保存するために B_{j0}, B_{j1} の2つを用いたが，ブロードキャストの速度が遅い場合には，より多くの複製を保持する必要がある．

また，本アルゴリズムではプロセッサ PE_j 中の B_{j0} が非常に良い個体であった場合，必要以上にコピーされることを防ぐために B_{j0} がコピーされた回数を各プロセッサにブロードキャストし，適応度に対応する回数以上コピーされることを防いでいるが，複数個のプロセッサが全く同時に選択処理を始めた場合には，同一の個体が複数のプロセッサからコピーされることが起こり得る．このような必要以上の複製は探索全体を局所解へと導くが，評価結果を見る限りこのような問題は起きていない．これは，選択処理の時間が他の処理に比べて十分短いためである（MC では数千個のノードの移動を繰り返し行うのに対し，選択では高々数十個の個体の適応度を計算するだけである）．

4.1.2 スケラビリティ

このアルゴリズムのスケラビリティは，ブロードキャストの速度によって定まる．しかし，文献 10, 14 に報告されているように，個体を複数のコロニーに分割し，各コロニー間で適当に個体の交換を行うことによって，全個体をひとまとめとして扱った場合と同程度の性能を実現することができる．本アルゴリズムをこの種の手法と組み合わせることによって，より高並列な処理を実現することができる．

4.2 問題向きオペレータ

交叉および突然変異においては，適度に個体を変更

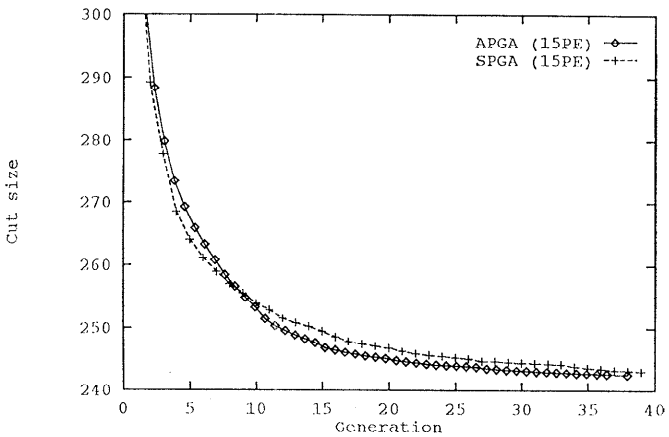


図 2 リンク数 (test 4)
Fig. 2 Cut size (test 4).

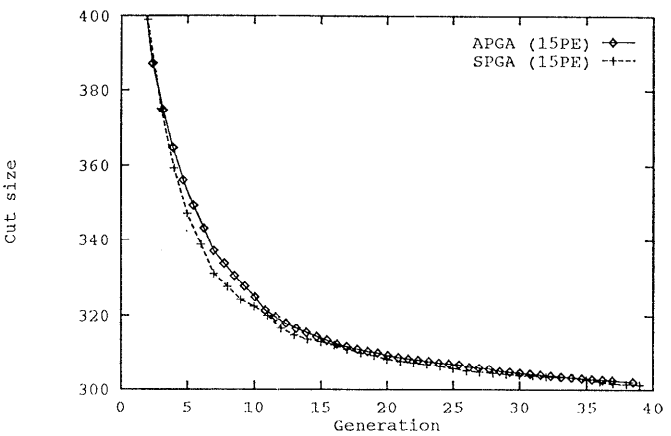


図 3 リンク数 (test 5)
Fig. 3 Cut size (test 5).

することが要求される．過度に個体の変更を行うとかえって個体の評価値が悪くなり，変更が少なすぎるとローカルミニマムに陥る可能性が高くなるからである．

4.2.1 交叉オペレータ

本アルゴリズムでは，各個体は“01”からなるビット列で構成されている．各ビットはそれぞれ1つのノードに対応しており，“0”はそのノードがグループ0に，“1”はグループ1に所属していることを表す．交叉は2つの個体 $I[j]$, $B[j]$($j=1, L$; L は個体の長さ(ノード数)) に対して以下のように行われる．

1. L の約 CX% 程度の長さの部分列を $B[j]$ から切り出す（切り出す長さは乱数で決められる．本論文の評価では約 40% を切り出している）．部分列の数は1または2であり（これもランダムに決められる），その合計が L の約 CX% となるように選ばれる．
2. 切り出した部分列を $I[j]$ の対応する位置に埋め込む．

この処理によって，個体 $I[j]$ 中の CX で指定された割合のノードのグループ番号が $B[j]$ のグループ番号で置き換えられる．しかし，交叉の結果生成された個体は一般にグループ内のノード数（ノードの重さ）に関する制約を満たさない．このため，幾つかのノードをグループ間で移動して各グループに割り振られるべきノード数（重さ）に関する制約を満たすように調整する必要がある．例えばすべてのノードの重みが1であるとした場合には，2つのグループの重みの差が1以下になるように，2つのグループ間でノードの移動を行わなければならない．この制約を満たすようにノードの移動を行う場合に以下の2種類の処理を確率的に選択して実行している．以下，$W(G_0)$, $W(G_1)$ は，それぞれグループ G_0, G_1 中のノードの重みの合計を表すものとし，$W(G_0) > W(G_1)$ であるとする．

1. 確率的にノードを選択して移動
   $W(G_0) > W(G_1)$ である間，G_0 中のあるノードを1つランダムに選択し，G_1 に移動する．
2. よりグループ間のリンク数の多いノードを移動
   (a) G_0 中の各ノード N_i を $GAIN(N_i)$ によってソートする．ただし
   $GAIN(N_i)=$
   $\sum(N_i$ から G_1 中のノードへのリンク数$)-$
   $\sum(N_i$ から G_0 中のノードへのリンク数$)$
   (b) $W(G_0) > W(G_1)$ である間，GAIN の大きなノードから G_1 に移動する．

前者のみだと解の収束速度が遅くなるが，後者のみであると局所解に陥る危険性が高くなる．このため両者を組み合わせて用いている．後者の処理は，実際には

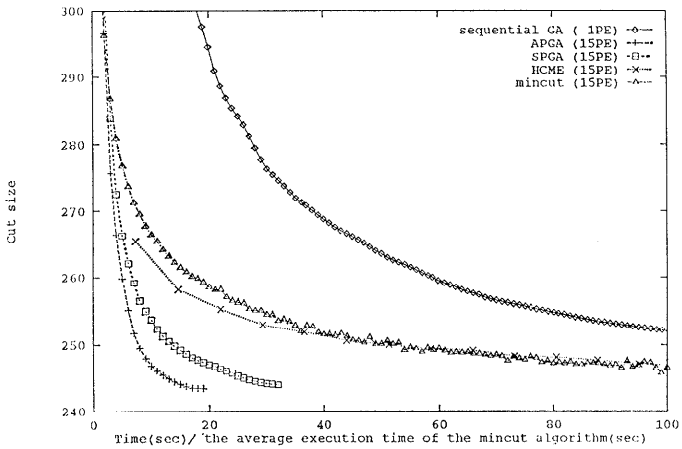


図 4 リンク数 (test 4)
Fig. 4 Cut size (test 4).

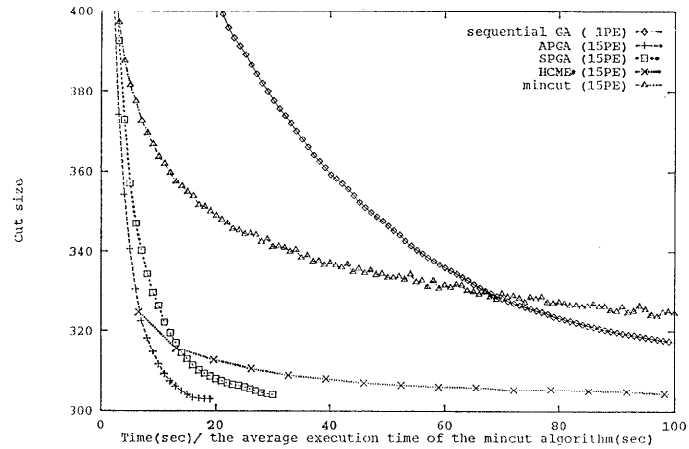


図 5 リンク数 (test 5)
Fig. 5 Cut size (test 5).

MC の一部と全く同じ処理である．

4.2.2 突然変異オペレータ

一般の GA では，突然変異オペレータは，個体のごくわずかの部分（通常1ビット分）しか変更しない．しかし，MC と組み合わせた場合には，このような小規模の変更は，MC の適用によって直ちに元の局所解へと復元されてしまう．このため，局所解から突然変異によって抜け出すためには，より多くの部分を変更することが必要となる（評価では約 20% 程度のノードをグループ 0，1 間で入れ換えている）．この結果，MC と組み合わせた場合には，突然変異と交叉の差異は，それほど明確ではなくなる．このため，各世代ごとにどちらかのみを適用している．

5. 評　　価

実際のゲートアレイの回路から得られた6種類のグラフの2分割について，以下に示す4種類のアルゴリズムの評価を行った．いずれの評価においても 15 プロセッサを用いている．なお，本論文中の評価結果は，アルゴリズムとしてのより厳密な比較を行うためにすべてC言語で記述されたものを用いている．また，各アルゴリズムは，基本的には MC に基づくものであり，各プログラムとも同一の関数を用いているため，コーディング等による性能の差はほとんどない．

1. Asynchronous PGA (Parallel GA)
2. Synchronous PGA (Parallel GA)
3. MC (15 PE)
4. HCME (15 PE)

Asynchronous PGA は本論文で提案を行っているアルゴリズムである（以下，APGA と略す）．Synchronous PGA は交叉，選択において従来の GA の方法を用いたものである（以下 SPGA と略す）．これらの評価においては，交叉オペレータでは約 40% のノードが置き換えられており，突然変異オペレータでは約 20% のノードの変更が行われている．交叉オペレータと突然変異オペレータを適用する比率は 7：3 で固定である．2種類の交叉オペレータの比は 1：1 である．

MC (15 PE) は，15 台のプロセッサで MC を独立に実行した場合の最良値である．MC は1回の処理に要する時間が短いため初期値を変えながら何回か繰り返し行っている．

HCME は MC と比べて約 5〜7 倍の処理時間を必要とする．HCME (15 PE) は 15 台のプロセッサを用いて MC (15 PE) と同様の処理を行って最も良かった値である．

以上の評価においては，グラフのデータをロードする時間は含まれていない．また，各評価値は 40 回の試行の平均値である．

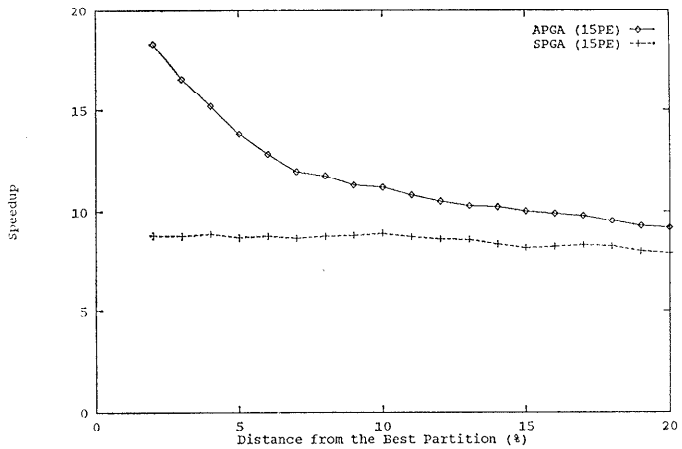


図 6 速度向上率 (test 4)
Fig. 6 Speedup (test 4).

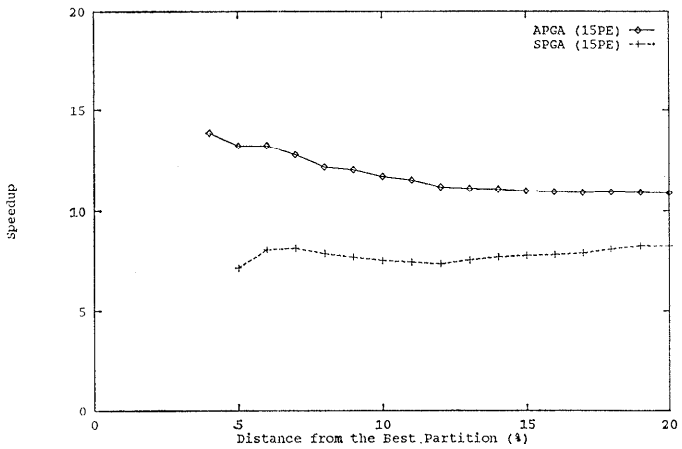


図 7 速度向上率 (test 5)
Fig. 7 Speedup (test 5).

5.1 探索能力

各プロセッサにおいて，HCME 2回分の時間を経過した時点でのリンク数を**表1**に示す．従って，表1における HCME の値は 30 回の最良値である（2×15 PE＝計 30 回）．HCME は1回の実行に MC の 5〜7 倍の処理時間がかかるため表1において MC は，約 150〜220 回の最良値となっている．すべての例題において，AGPA が最も良い値を示していることが分かる．APGA および SPGA における遺伝的アルゴリズム固有の処理時間は，全処理時間の 10% 程度である．

次に，APGA の遺伝的アルゴリズムとしての探索能力を評価するために例題 test 4 と test 5 における，APGA と SPGA の世代ごとのリンク数を**図 2, 3** に示す．図2，3において縦軸は評価値（グループ間のリンク数）であり少ないほど良い．横軸は遺伝的アルゴリズムにおける世代数である．APGA においては各処理が非同期化されているため世代という概念はないが，SPGA と同一回数の Mincut アルゴリズムが適用された時点でのリンク数を表示した．図 2，3から，非同期化によって遺伝的アルゴリズムとしての探索能力は全く損なわれていないことが分かる．

5.2 探索速度

例題 test 4 および test 5 における，探索の収束速度を**図 4, 5** に示す．Sequential GA は，SPGA を1台のプロセッサで実行した場合のものである．縦軸は評価値（グループ間に跨るリンク数）である．横軸は実行時間であり，MC 1回の平均実行時間で正規化されている．APGA が非常に速い収束を見せていることが分かる．

5.3 並列度

図 6, 7 に，図4, 5における APGA と SPGA の並列処理による速度向上率を示す．これは，これまでに得られている最も良い評価値からある程度の範囲内の値（X軸に範囲が示されている）を見つけるまでの処理時間を比較したものである．15プロセッサを用いて最大 14〜18 倍の高速化が実現されている．

この非常に良い速度向上率の可能性としては以下の2種類が考えられる．

- プロセッサ数の増加による実質的なキャッシュメモリサイズの増大
- より良い評価値を持つ個体には単位時間内により多

表 1 各アルゴリズムによるリンク数（40回の平均）
Table 1 Cut size (average of 40 runs).

Example	ノード数	APGA	SPGA	HCME	MC
test 1	249	36.0	37.5	37.5	36.0
test 2	615	71.0	71.5	71.4	81.7
test 3	1861	254.6	258.6	270.0	274.6
test 4	1869	244.2	249.3	258.2	261.7
test 5	3096	306.2	316.7	315.7	357.1
test 6	3373	186.2	186.0	188.0	207.3

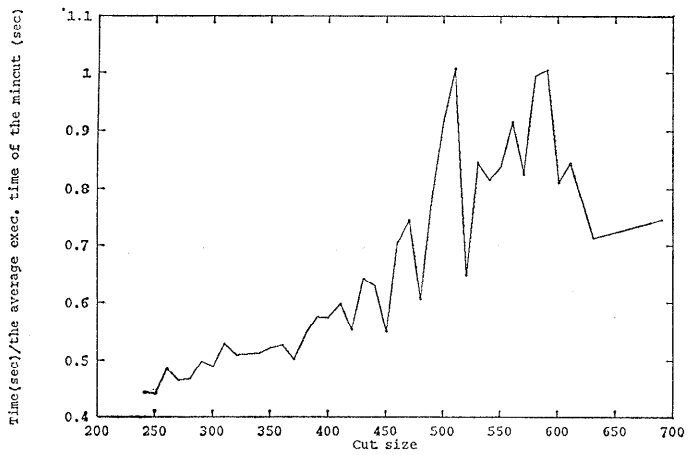


図 8 MC の適用に要する処理時間 (test 4)
Fig. 8 The average time required for applying the mincut algorithm (test 4).

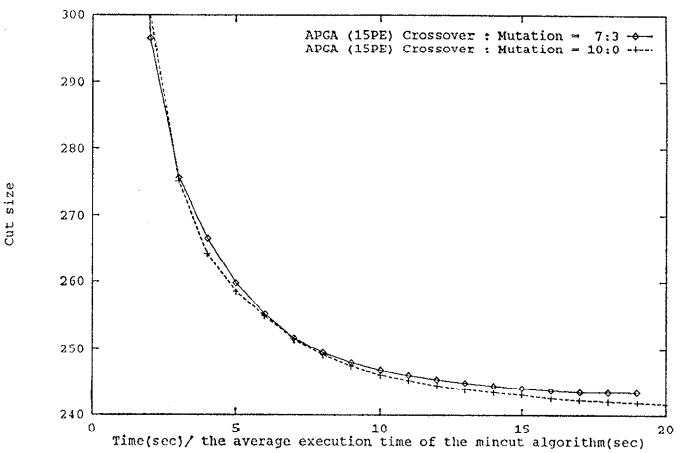


図 9 パラメータの影響 (test 4)
Fig. 9 Parameters (test 4).

くの MC が適用可能

前者の影響については不明であり，今後の検討課題である．

図 8 に MC を 1 回適用するのに要した処理時間を示す．図 8 において縦軸は 1 回の MC に要した処理時間（MC 1 回の平均処理時間で正規化），横軸は MC を適用した後の評価値である．縦軸を正規化するための MC 1 回の平均処理時間には，図 4，5 と同じ指標を与えるために，初期化（乱数を用いたグラフの 2 分割，各種データの初期化等）を含むものを用いている．GA においては繰り返し MC の適用が行われるため，初期化等が不要であるので全般的に処理時間が短くなっている．

図 8 から分かるように評価値によって MC の適用に要する処理時間が大きく異なり，評価値 250 程度にいる個体は，評価値 500 程度にいる個体の約 2 倍の速度で MC が適用されることが分かる．このため，本アルゴリズムのような非同期処理を行った場合，より良い解はさらにより多くの MC の適用を受け，さらに良くなると考えられる．これに対して逐次型アルゴリズムではすべての個体が同一回数の MC の適用を受けている．従って本アルゴリズムでは最も良い個体は，同一時間内に逐次型アルゴリズムの最大 2 倍の回数の MC の適用を受けている可能性がある．このため，図 6，7 に示したように，より良い解を求めようとするに従って，この影響が表れ，プロセッサ台数を越える速度向上率が得られることがある．

5.4 パラメータ

図 9 と **10** に，例題 test 4 と test 5 において交叉オペレータと突然変異オペレータの適用比率を変えた場合の収束速度を示す．図 9 では，交叉オペレータのみの場合の方が良い結果が得られているが，図 10 では突然変異オペレータも用いた方が良い結果が得られている．実際には各例題ごとに適切なパラメータは異なり，すべての問題において良い結果が得られるようなパラメータを見つけることは困難であると思われる．

5.5 選択方式

表 2 に，同じく非同期型の遺伝的アルゴリズムであ

表 2 選択方式によるリンク数の変化（40 回の平均）
Table 2 Cut size after the same number of the mincut algorithm is applied (average of 40 runs).

Examples	APGA	APGA-LS	SPGA
test 1	36.0	36.0	36.0
test 2	71.0	71.0	71.0
test 3	254.0	257.7	254.8
test 4	243.4	247.2	243.9
test 5	302.9	319.1	304.1
test 6	186.2	187.2	186.2

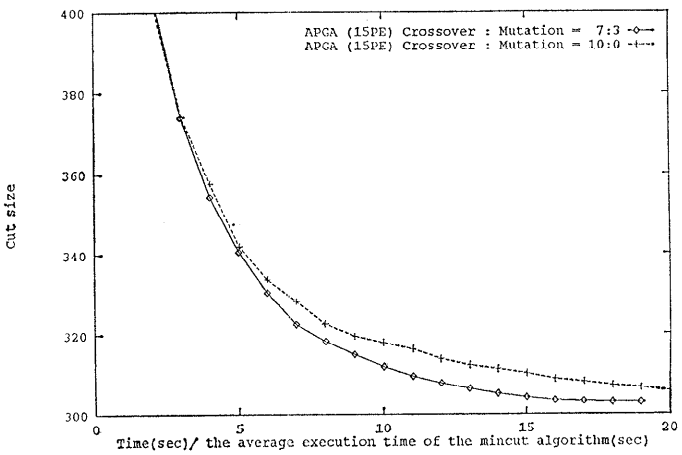


図 10 パラメータの影響 (test 5)
Fig. 10 Parameters (test 5).

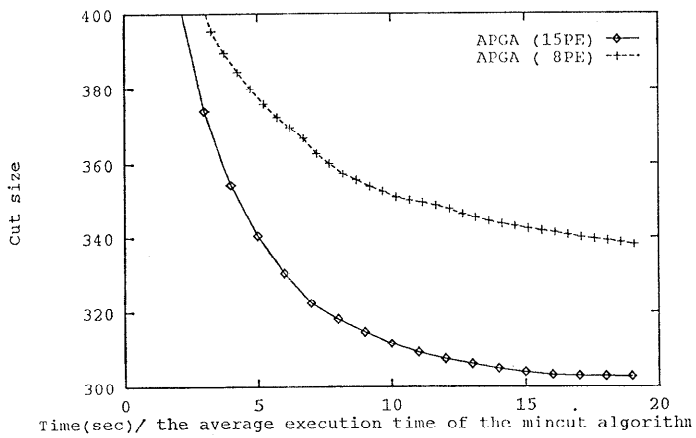


図 11 リンク数 (test 5)
Fig. 11 Cut size (test 5).

る文献 9 における選択方式を用いた場合（APGS-LS）と APGA, SPGA との比較を示す．表2は，同一回数（480 回＝15×32 世代相当分）の MC が適用された時点での値である．文献 9 の選択方式では，問題の規模が小さな場合には SPGA と同程度の探索性能を示すが，問題規模が大きな場合には，次第に探索能力が低下することが分かる．

5.6 スケラビリティ

図 11 に test 5 において，個体数を 8 個とした場合と 15 個の場合の比較を示す．この結果から分かるように良い結果を得るためにはある程度以上の個体数が必要である．しかし，APGA はブロードキャストに基づくアルゴリズムであるため，ある程度以上の個体数を同時に扱うことができない．この上限は並列マシンのブロードキャスト等の性能によって決まる．しかし，第 4.1.2 節で述べたように，複数個のコロニーを用いることによって，全体をひとまとめとして扱った場合と同程度の性能を期待することができる．

図 12 に，複数個のコロニーを用いて例題 test 5 の探索を行った場合の結果を示す．この評価は，十分なプロセッサ数を持つ並列マシンを用いることができなかったため，同期型並列 GA（SPGA）を用いて行った．図 12 は，16 個のコロニーをリング上に配置し，各コロニーには 16 個の個体を配置した場合のものである．従って，各コロニーは交換確率に応じて両隣のコロニーと個体を交換する（図 12 では各世代または 2 世代ごとに隣接するコロニー間で 1 個体を交換）．複数個のコロニーに分割しても，全体を 1 つのコロニーとして扱った場合と同程度の性能が実現されていることが分かる．従って，本非同期型細粒度並列遺伝的アルゴリズムを用いてより高並列な処理を実現することができる．

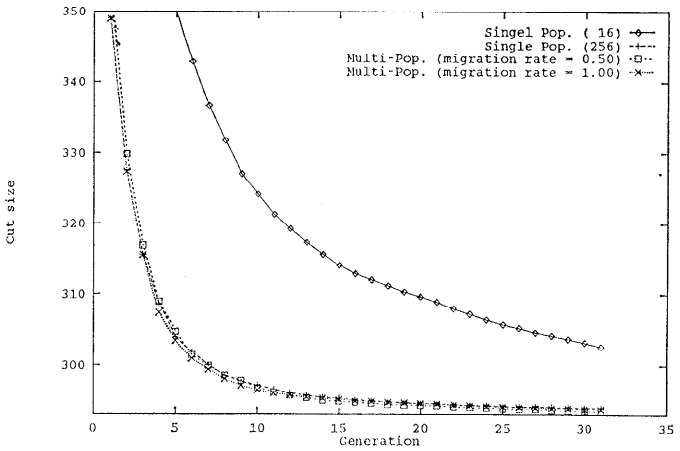


図 12 複数のコロニーによる探索 (test 5)
Fig. 12 Search by multiple population strategy (test 5).

6. おわりに

本論文では，遺伝的アルゴリズムを用いた新たな並列グラフ分割アルゴリズムを提案し，このアルゴリズムを用いることによって，従来のヒューリスティックアルゴリズムと比べてより良い解をより高速に探索することが可能であることを示した．この並列グラフ分割アルゴリズムは現在 Symmetry 上に実装されており，15 プロセッサを用いて最大 14〜18 倍程度の性能向上率を実現している．

また，本論文で提案する細粒度並列遺伝的アルゴリズムと，複数のコロニーを用いる粗粒度並列遺伝的アルゴリズムを組み合わせることによって，より高並列な処理が実現できることをシミュレーションにより示した．

今後，より多くのプロセッサからなる並列システムに本アルゴリズムを実装し，本アルゴリズムの並列性の評価を行う予定である．

謝辞 この研究の場を用意してくださった ICOT 内田研究部長，近山室長に感謝いたします．

参考文献

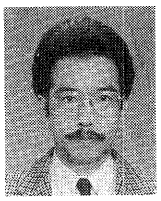
1) Kernighan, B. W. and Lin, S.: An Efficient Heuristic Procedure for Partitioning Graphs, *Bell Systems Technical Journal*, pp. 291-307 (1970).
2) Fiduccia, C. M. and Mattheyses, R. M.: A Linear Time Heuristics for Improving Network Partitions, *ACM/IEEE Design Automation Conf.*, pp. 175-181 (1982).
3) Edahiro, M. and Yoshimura, T.: New Placement and Global Routing Algorithms for Standard Cell Layouts, *Proc. of 27th DAC*, pp. 642-645 (1990).
4) Wei, Y. C. and Cheng, C. K.: Towards Efficient Hierarchical Designs by Ratio Cut Partitioning, *IEEE Intl. Conf. on Computer-Aided Design*, pp. 298-301 (1989).
5) Hagen, L. and Kahng, A.: Fast Spectral

Methods for Ratio Cut Partitioning and Clustering, *IEEE Intl. Conf. on Computer-Aided Design*, pp. 10-13 (1991).
6) Kring, C. and Newton, A. R.: A Cell-Replicating Approach to Mincut-Based Circuit Partitioning, *IEEE Intl. Conf. on Computer-Aided Design*, pp. 2-5 (1991).
7) Goldberg, D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley (1989).
8) Kitano, H., Smith, S. F. and Higuchi, T.: A Parallel Associative Memory Processor for Rule Learning with Genetic Algorithms, *Proc. of Intl. Conf. on Genetic Algorithms*, pp. 311-317 (1991).
9) Gorges-Schleuter, M.: ASPARAGAS An Asynchronous Parallel Genetic Optimization Strategy, *Proc. of Intl. Conf. on Genetic Algorithms*, pp. 422-427 (1989).
10) Muhlenbein, H., Schomish, M. and Born, J.: The Parallel Genetic Algorithm as Function Optimizer, *Proc. of Intl. Conf. on Genetic Algorithms*, pp. 271-278 (1991).
11) Spiessen, P. and Manderick, B.: A Massively Parallel Genetic Algorithm—Implementation and First Analysis, *Proc. of Intl. Conf. on Genetic Algorithms*, pp. 279-286 (1991).
12) von Laszewski, Gv.: Intelligent Structural Operators for the k-way Graph Partitioning Problem, *Proc. of Intl. Conf. on Genetic Algorithms*, pp. 291-307 (1991).
13) Collins, R. J. and Jefferson, D. R.: Selection in Massively Parallel Genetic Alogorithms, *Proc. of Intl. Conf. on Genetic Algorithms*, pp. 249-256 (1991).
14) Cohoon, J. P., Martin, W. N. and Richards, D. S.: A Multi-population Genetic Algorithm for Solving the K-Partitioning Problem on Hypercubes, *Proc. of Intl. Conf. on Genetic Algorithms*, pp. 244-248 (1991).
15) Konishi, K., Maruyama, T., Konagaya, A., Yoshida, K. and Chikayama, T.: Implementing Streams on Parallel Machines with Distributed Memory, *Proc. of Intl. Conf. on Fifth Generation Computer Systems*, pp. 791-798 (1992).

(平成4年9月14日受付)
(平成5年1月18日採録)

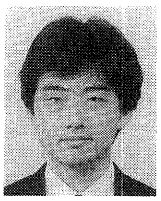
丸山 勉（正会員）

1982年東京大学工学部電子工学科卒業．1987年同大学院情報工学専門課程博士課程修了．同年日本電気株式会社に入社．以来，並列処理，分散処理のためのプログラミング言語および処理系，並列アルゴリズムの研究に従事．現在C&Cシステム研究所コンピュータシステム研究部主任．



小長谷明彦（正会員）

1978年東京工業大学理学部情報科学科卒業．1980年同大学院修士課程修了．同年日本電気株式会社に入社．C&Cシステム研究所において並列処理システムの研究に従事．1987年より遺伝子情報処理，1990年より遺伝子アルゴリズムの研究を進めている．現在，コンピュータシステム研究部研究課長．電子情報通信学会，ソフトウェア科学会，人工知能学会の会員．



小西 弘一（正会員）

1987年東京大学工学部計数工学科卒業．1989年同大学院修士課程情報工学専攻修了．同年日本電気株式会社に入社．以来，並列処理，分散処理のためのプログラミング言語および処理系の研究に従事．現在C&Cシステム研究所コンピュータシステム研究部に所属．