

TM-1188

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 (日電)

June, 1992

© 1992, ICOT

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

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

日本電気(株) C&C システム研究所

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較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、非同同期型細粒度並列遺伝的アルゴリズム及び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。

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.

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較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、非同同期型細粒度並列 GA を提案し、同期型の並列 GA と較べて約 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。

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[12] の評価の一環として行なわれたものである。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な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(レイアウト等)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 [1, 2] 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ムに陥り十分によい解を得ることができない。一方、Simulated Annealing は、よい解を求めることができるようにチューニング可能であるが、一般にその処理時間はかなり大きい。今後回路の大規模化が進むことを考えるとより高速なグラフ分割アルゴリズムが必要である。

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なう確率的探索アルゴリズムである [4]。この一回分の処理を世代と呼ぶ。

- 交叉 (Crossover)
- 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なわれ、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。選択も確率的に行なわれ、比較

的悪い評価値を持つ個体も、その評価値に応じた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なうため、ある程度の値には比較的容易に近づくが(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ない、グローバルな山登りに GA を利用することによって非常によい解を得ることができる。

3.2 並列遺伝的アルゴリズム

GA は複数個の個体を用いて探索を行なうため、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 [5, 6, 7, 8, 9, 10, 11]。

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、コロニー間で適度に個体を交換し合うことによって探索を行なう。即ち、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なう。[7, 11] では、コロニー間で適度に個体の交換を行なうことによって、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プロセッサが担当する。従って、交叉、選択は通信のオーバーヘッドを軽減するために近傍のプロセッサ間のみで行なわれる。粗粒度並列処理では選択を非同期化した方式 [6] が提案されている。[6] 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、そうでな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。

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

本アルゴリズムでは、ヒューリスティックアルゴリズムとして 1-opt Mincut アルゴリズム ([1, 2] 参照) を用いている (以下、MC と呼ぶ)。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 の概要について述べる。次いで、本アルゴリズムのクリティカルセクション及びスケラビリティについて述べる。

アルゴリズム概略

GA において細粒度の並列化を行な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。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毎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、以下のように各処理を非同期化することによって、待ち合わせを極力排除している。

1. 各プロセッサに 1 個体及びそのコピーを格納するバッファを設ける。
2. 各個体のコピーをバッファに格納する。
3. 各個体の評価値をブロードキャストする。
4. 各個体について突然変異、またはバッファ中のコピーとの交叉を行なう。

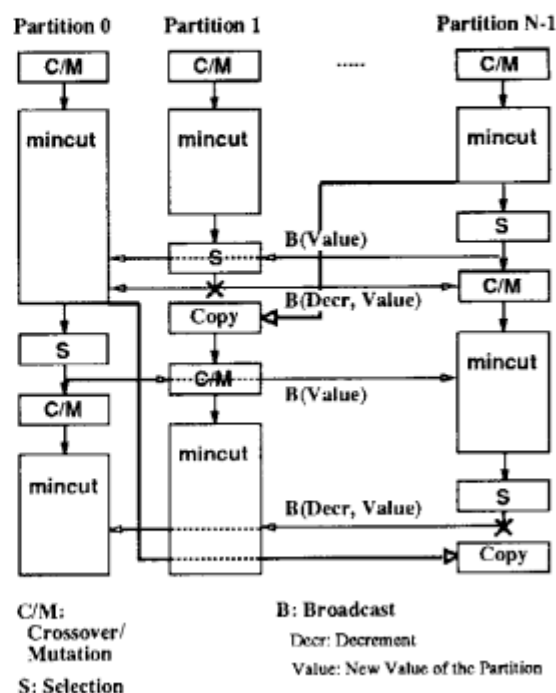


図 1: 非同期型細粒度並列 GA 概略

5. 各個体について MC を適用する

6. 各個体について、ブロードキャストされた評価値を用いて選択を行なう。

- 各プロセッサ中の個体が生き残った場合には、その個体の評価値をブロードキャストし、その個体のコピーをバッファに格納する。
- 生き残れなかった場合には、ブロードキャストされている評価値に応じて、他のプロセッサ中のバッファを選択してコピーし、新たにそのプロセッサの個体とする。この時、新たな個体の評価値として十分に悪い評価値をブロードキャストする。また、選択されたバッファ中のコピーの評価値をブロードキャストによってある程度減ずる。

7. 4~6の処理を各個体について繰り返す

アルゴリズムの概略を図 1 に示す。

交叉については、従来の交叉のように2つの個体から2つの子孫を生成するのではなく、1つの個体の一部をバッファ中のコピーの1つの一部で置き換えることによって1つの子孫を生成する。このようにして2個体間での間の待ち合わせを防いでいる。

選択に関しては、各個体の評価値をバッファ中のコピーの評価値と比較し、良い場合にはその個体のコピーをバッファに格納し、そうでない場合には他のプロセッサのバッファ中のコピーから1つを選択し(より良い評価値をもつ個体をより高い確率で選択する)、コピーして、それをそのプロセッサの個体とする。一度選択されたコピーの評価値は一定量減じて(ブロードキャストによって行なわれる)、同じ個体が何回も選択されることを防ぐ。このようにすることによって、選択における待ち合わせを排除することができる。

このような非同期化による利点は、待ち時間の排除以外にも次のものが考えられる。MCでは、一般に解が良くなるに従ってその処理時間が短くなる。このため、より良い個体は単位時間内により多くのMCの適用を受けることができ、さらに良い解が得ることが期待できる。

クリティカルセクション

この非同期型並列GAには、3つのクリティカルセクションがある。ひとつは、バッファの更新である。実際には、バッファの更新はポインタの更新処理のみでよいためクリティカルな部分の処理時間は極めて短い。このため、バッファの更新時にはバッファをロックしている。さらに、交叉のためにバッファ中のコピーが読み出されている最中でも、そのコピーに関する更新が可能なようにバッファを2面用いている。

他の2つは、ブロードキャストによる評価値の更新に伴うものである。まず、 PE_i 中の個体が選択において生き残れなかったため、 PE_i は新たな個体として PE_j 中のバッファをコピーするものとする。この PE_i での選択の最中に、 PE_j 中の個体が PE_j での選択の結果生き残り、その評価値の更新が PE_j によって行なわれていると、 PE_i は、 PE_j によって更新されたばかりの PE_j 中の個体の評価値を

誤って一定量減じてしまうことになる。実際の処理系では、このような場合には、一定量減ずるためのブロードキャストは抑止される。

次に、 PE_i と PE_j の2つのプロセッサが同時に選択を開始し、 PE_i, PE_j 中の個体が双方ともに生き残れなかったとする。このとき、 PE_k 中のバッファのコピーの評価値が良い場合には、このコピーが PE_i, PE_j の両方にコピーされることがある(PE_i の選択処理が先に終了し、 PE_k 中のコピーの評価値が一定量既に減じられていれば、このようなことは起こらない)。このような過度のコピーは、局所解に陥る危険性を高めるが、選択処理の処理時間は、他の処理に較べて非常に短いため、ブロードキャストの速度が十分速ければこのような可能性はほとんど問題とはならない。

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、[7, 11]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なうことによって、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

交叉及び突然変異においては、適度に個体を変更することが要求される。過度に個体の変更を行なうと返って個体の評価値が悪くなり、変更が少な過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。

交叉オペレータ

交叉の結果生成された子孫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。この制約を満たすようにノードの割り当てを変更する場合に以下の2種類の処理を確率的に選択して実

行している。

- 確率的にノードを選択して移動
- よりグループ間に跨るリンク数の多いノードを優先的に移動

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。

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、個体のごく僅かの部分しか変更しない。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。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なった。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、本論文中的評価結果は、アルゴリズムとしてのより厳密な比較を行なうために全て C 言語で記述されたバージョンを用いている。

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

Asynchronous PGA は本論文で提案を行な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%、突然変異

オペレータでは約 20% のノードの割り当てが変更されている。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(15PE) 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 は一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なっている。

HCME[3] は、MC を改良したヒューリスティックアルゴリズムである。HCME は MC と較べて約 5~7 倍の処理時間を必要とする。HCME(15PE) は 15 台のプロセッサを用いて MC(15PE) と同様の処理を行なって最も良かった値を表示したものである。HCME はノードのグループ化を行なった後、MC を適用するアルゴリズムであるが、グループ化において選択肢が多数存在するため、それらを適当に選択することによって異なる結果となる。

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。

5.1 探索能力

各プロセッサにおいて、HCME が 2 回終了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 ($2 \times 15PE =$ 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。全ての例題において、APGA が最も良い値を示していることが分かる。

5.2 探索速度

例題 test4 及び test5 における、探索の収束速度を図 2,3 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 一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、及び SPGA と結果的には同程度の解を見つけており、非同期化によって解の質が損なわれていないことが分かる。

表 1: 各アルゴリズムによるリンク数 (40 回の平均)

Example	#cells	APGA	SPGA	HCME	mincut
test1	249	36.0	37.5	37.5	36.0
test2	615	71.0	71.5	71.4	81.7
test3	1861	254.6	258.6	270.0	274.6
test4	1869	244.2	249.3	258.2	261.7
test5	3096	306.2	316.7	315.7	357.1
test6	3373	186.2	187.0	188.0	207.3

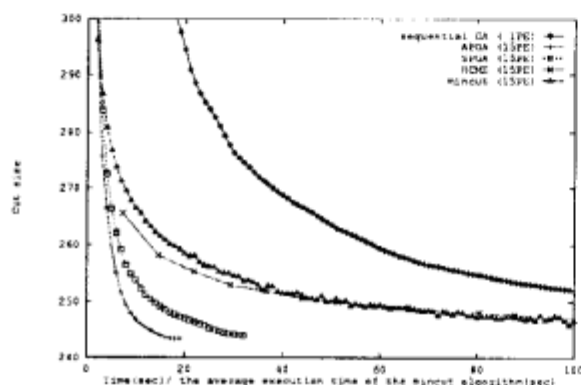


図 2: リンク数 (test4)

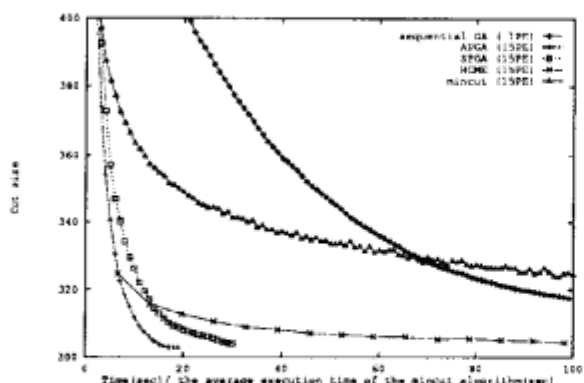


図 3: リンク数 (test5)

5.3 並列度

図 4,5に、図 2,3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 $14 \sim 18$ 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アイズの増大
- より良い値は単位時間内により多くの 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 6に MC を 1 回適用するのに要した処理時間を示す。図 6において縦軸は 1 回の MC に要した処理時間 (MC1 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。図 6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なった場合、より良い解はさらにより多くの処理を受け、さらに良くなると考えられる。このため、図 4,5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、より高い速度向上率が得られている。

5.4 パラメータ

図 7と 8に、例題 test4 と test5 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 7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 8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。実際には各例題毎に適切なパラメータは異なり、全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。

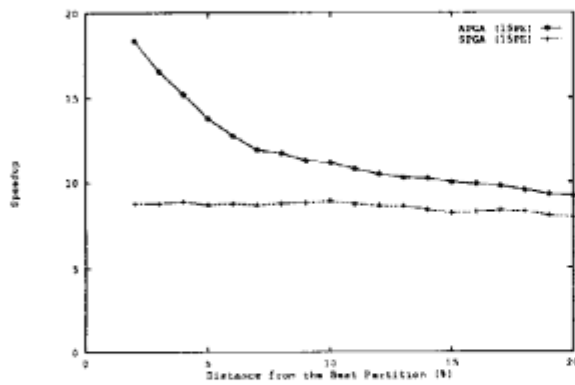


図 4: 速度向上率 (test4)

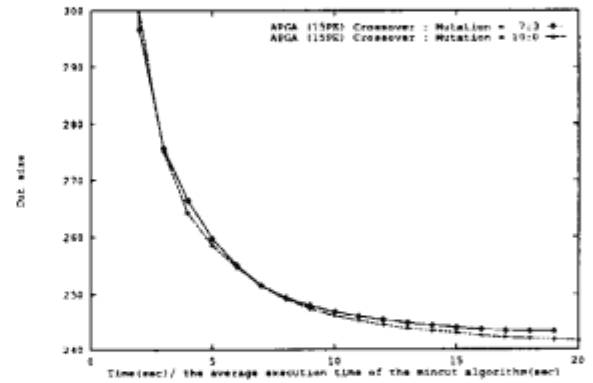


図 7: パラメータの影響 (test4)

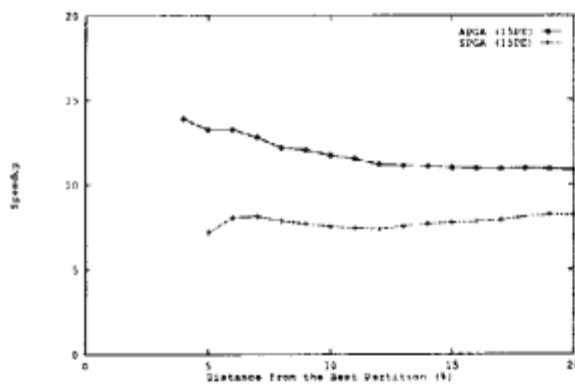


図 5: 速度向上率 (test5)

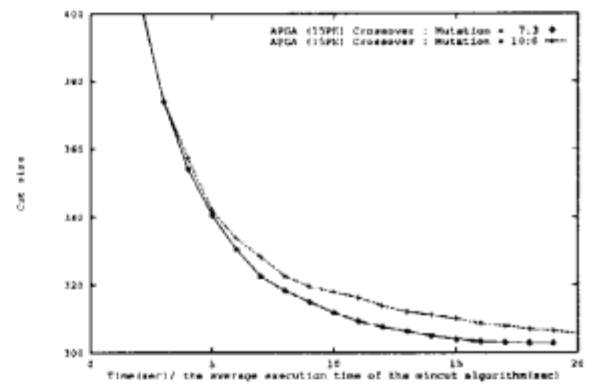


図 8: パラメータの影響 (test5)

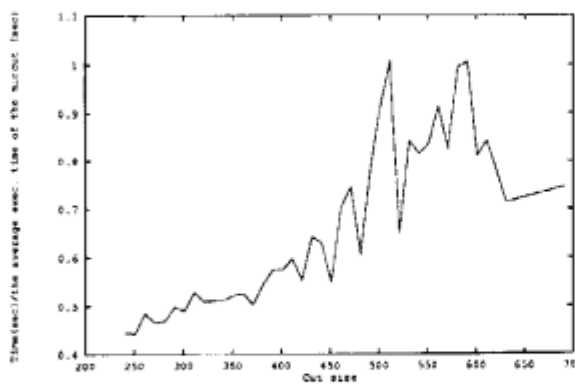


図 6: MC の適用に要する処理時間 (test4)

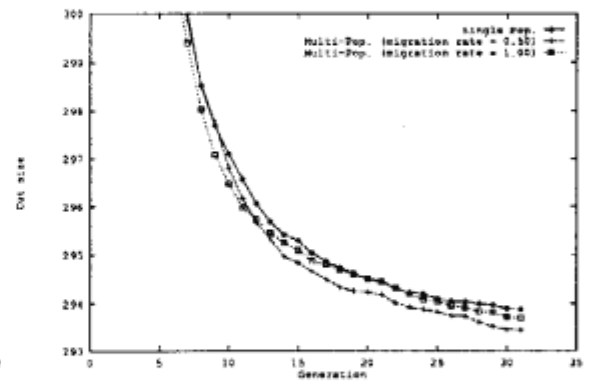


図 9: 複数のコロニーによる探索 (test5)

5.5 スケラビリティ

図9に、複数のコロニーを用いた場合の評価を示す。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)を用いて行なった。図9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隣のコロニーと個体を交換する(図9では各世代または2世代毎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わかる。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。

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較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。

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、近山室長に感謝致します。

参考文献

- [1] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", Bell Systems Technical Journal, 1970, pp.291-307.
- [2] C. M. Fiduccia and R. M. Mattheyses "A linear Time Heuristics for Improving Network Partitions", ACM/IEEE design Automation Conf., 1982, pp.175-181.
- [3] M. Edahiro and T. Yoshimura, "New Placement and Global Routing Algorithms for Standard Cell Layouts", Proc. of 27th DAC, 1990, pp 642-645.
- [4] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1989.
- [5] H. Kitano, S. F. Smith and T. Higuchi, "A parallel Associative Memory Processor for Rule Learning with Genetic Algorithms", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 311-317.
- [6] M. Gorges-Schleuter, "ASPARAGAS An Asynchronous Parallel Genetic Optimization Strategy", Proc. of Intl. Conf. on Genetic Algorithm, pp 422-427.
- [7] H. Muhlenbein, M. Schomish and J. Born, "The Parallel Genetic Algorithm as Function Optimizer", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 271-278.
- [8] P. Spiessen and B. Manderick, "A Massively Parallel Genetic Algorithm - implementation and First Analysis", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 279-286.
- [9] G. von Laszewski, "Intelligent Structural Operators for the k-way Graph Partitioning Problem", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 291-307.
- [10] R. J. Collins and D. R. Jefferson, "Selection in Massively Parallel Genetic Algorithms", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 249-256.
- [11] J. P. Cohoon, W. N. Martin and D. S. Richards, "A Multi-population Genetic Algorithm for Solving the K-Partitioning Problem on Hypercubes", Proc. of Intl. Conf. on Genetic Algorithm, 1991, pp 244-248.
- [12] K. Konishi, T. Maruyama, A. Konagaya, K. Yoshida and T. Chikayama, "Implementing Streams on Parallel Machines with Distributed Memory", Proc. of Intl. Conf. on Fifth Generation Computer Systems, 1992.