成蹊大学工学研究報告
J. Fac. Eng., Seikei Univ.
Vol.40 No.2 (2003) pp.35-41

並列化ＧＡによる通信時間を考慮したタスクスケジューリング解法

時村良平*1，甲斐宗徳*2

Solving task scheduling problems taking account of communication overhead using parallelized GA

Ryouhei TOKIMURA*1, Munenori KAI*2

ABSTRACT: In general, Genetic Algorithm(GA) has some faults in practical use. One of them is that it is difficult to construct a genetic representation of solutions for the target problem. Another is that it takes much processing time to give an optimal solution to any optimizing problems. In this paper, a set of effective genetic representations for the task scheduling problem is proposed, and in order to reduce the total processing time, we also propose a parallel processing scheme of GA process to solve the task scheduling problem. In the proposed scheme, there are multiple colonies, each of which has different genetic representations from each other and evolve independently or in parallel. Among the colonies, the best individuals are exchanged in every a few hundreds generations. By this parallel processing scheme, solutions of the task scheduling problem are given in the reduced processing time.

KEYWORDS: Task Scheduling, Optimizing Problem, Parallel Processing, Genetic Algorithm

(Received June 20, 2003)

1．はじめに

並列処理において十分な性能向上を得るためには，タスクスケジューリングを行うことが重要である。しかし，その最適解を得ることは事実上不可能であるがこと知られている。タスクスケジューリング問題のように，強NP困難で解法アルゴリズムを組むことのできない最適化問題や探索問題に対して有効な手法として，遺伝的アルゴリズム（Genetic Algorithm：以下GA）がある。このＧＡを利用してタスクスケジューリング問題の解を求める。また，GAは並列性の側面を強く持つアルゴリズムである。そこで，GAを並列化し，より良いスケジューリングの解を高速に求めることを目指す。

2．タスクスケジューリング問題

タスクとは，１つのプログラムを構成する部分要素であり，スケジューリング問題の対象となる１単位である。１つのプログラムは，複数のタスクからなるタスク集合とみなすことができる。このタスク集合を構成するタスク間には，実行順序の制約（先行制約）がある。各タスクは，タスク間で処理結果の受け渡しをしながら処理を進めていく。このため，処理を開始する前に先行タスクの処理結果を受け取らなければならない。このタスクの先行関係を表したものが“タスクグラフ”である。マルチプロセッサシステムのためのスケジューリング問題とは，コンパイラあるいはユーザにより抽出された並列実行可能なタスク集合を，複数のプロセッサにどのように割り当てて，どのような順序で実行させるかを決定することである。本研究では，ランダムに生成したタスクグラフを解くことで，タスクスケジューリング問題解法としての性能を評価する。

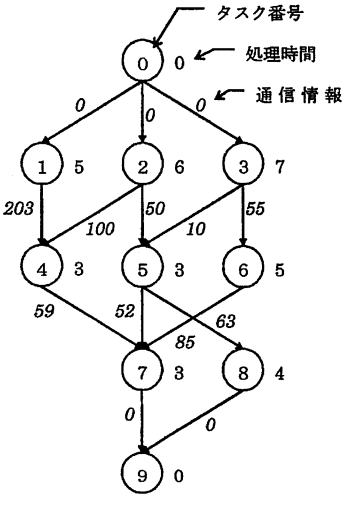


Fig. 1. タスクグラフ例

*1 情報処理専攻大学院生[現：ユニパルス㈱]
*2 情報処理専攻助教授(kai@is.seikei.ac.jp)
Associate Professor, Dept. of Information Sciences

3．GA

GAは自然界における生物の進化をモデルとしたアルゴリズムである。まず親から遺伝子によって生物としての情報伝達が行われる。次世代へは，各個体の中でもより優れた，つまり環境への適応度の高い個体の遺伝情報が優先的に伝えられる。適応度の低い個体は自然淘汰されていく。このような原理に基づいて世代進化を重ねていくと，次第に環境への適応度の高い個体が多くなっていく。これが遺伝と進化の基本的な原理である。Fig2にGAのフローチャートを示す。

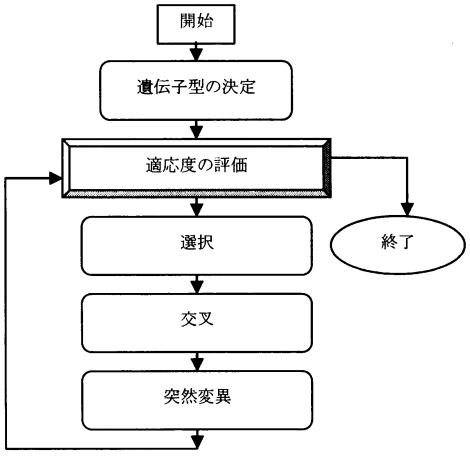


Fig. 2. GAアルゴリズムフローチャート

4．GA性能の改良

GA性能の改良に当たり，以下の手法を用いた。

4.1 GA解析ツール

GAの性能を改善するために，各染色体の変化過程をFig3のようにアニメーション表示させるようなツールを製作した。

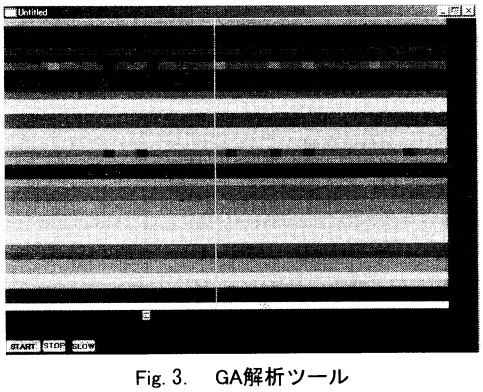


Fig. 3. GA解析ツール

横軸…各遺伝子(40個)
縦軸…各染色体(40個)
RGB(0, 0, 0)・・・各染色体のプロセッサ番号
色の明暗・・・各染色体のプライオリティ番号
(明・・・小，暗・・・大)
緑縦線…最良遺伝子
青横線…最良適応度

表示は世代数毎にアニメーションで変化する。このツールは簡易的ではあるが，視覚的にGA処理における染色体，遺伝子の変化の過程が解析できる。

4.2 遺伝子表現の改善

4.2.1 従来の遺伝子表現

従来の遺伝子表現では，プライオリティ番号と処理プロセッサ番号を使用していた。処理可能タスクのうち，もっともプライオリティの高いものを処理タスクとして選び，処理プロセッサは遺伝子表現の番号をそのまま使用する手法である。この手法ではタスクをひとつ処理する毎に割り当て可能タスクを更新していく必要がある。

Gene配列index(タスク番号)

	1	2	3	4	5	6	7	8
Gene[i].Priority	7	5	2	1	6	3	0	4
Gene[i].Processor	2	0	1	2	3	1	0	2

Fig.4. 従来の遺伝子表現

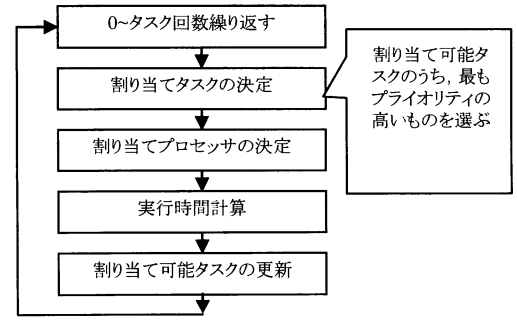


Fig. 5. 従来の評価(スケジューリング)方法

4.2.2 プライオリティ番号の無視

作成したツールで解析を行った結果，遺伝子表現のうち，プロセッサ番号の変化に比べて，プライオリティ番号の変化があまりないことがわかった。新たなコーディングとして，プロセッサ番号のみで評価をすることを考えた。

プライオリティ番号を使わずに，プロセッサ番号のみで評価するには，処理可能なタスクの順序をあらかじめ

計算しなければならない。そしてそのタスク順の表を毎回参照しながら割り当てタスクを決定していくことになる。

さらに，１交叉を行ったときの交叉点を記憶することで，高速化が可能になる。タスクの割り当て順が固定なので，遺伝子をタスク順に並べるのではなく，割り当て順にならべる。こうした場合，交叉点より前半の実行時間の計算に関しては，前世代の値を記憶しておけば省略できる。

この手法では，現在の割り当て可能タスクを考慮しなくてよいし，割り当てタスクの計算時間は表を参照するだけなので，計算時間はかなり早くなる(およそ1/2ほど)。しかし，処理タスク順は一度決めたら変化がないので，処理結果は偏ったものになる可能性がある。

Fig.6. プライオリティを無視する遺伝子表現と，処理タスク順表

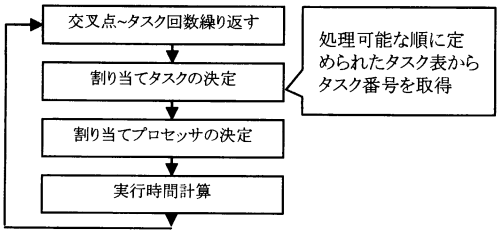


Fig. 7. プライオリティ番号を無視する評価(スケジューリング)方法

4.2.3 プロセッサ番号表現の工夫～従属割り当て～

遺伝子表現としてプロセッサ番号を直接表現するのでは効率が悪い。プロセッサコード表現に先行タスクと同一のプロセッサを使用するという表現をさらに付け加えてみた。この表現を使用することにより，通信時間の大きなタスク間を常に同一プロセッサで処理するように表現できる。

例えば，先行制約で結ばれたタスクが同一の処理プロセッサを持ち，通信時間がかからないという表現になっていたとする。従来のコーディングでは，交叉や突然変異でその関係はすぐに崩れていた。しかし，このコーディングでは先行タスクのプロセッサ番号に変化があったとしても，そのプロセッサ番号に従属した同じプロセッサを割り当てることになる。つまり，常に通信時間を0と表現できるというメリットがある。

この新しいプロセッサ番号の遺伝子表現を,「従属割り当て表現」と呼ぶことにする。遺伝子のうち，プロセッサ番号を直接指定したものと，この従属割り当て表現のいずれかでプロセッサ遺伝子を表現する手法を「従属割り当て法」と呼ぶことにする。従属割り当て法の実装では，プロセッサ遺伝子は,「従属割り当て表現をあらわす番号」と,「直接プロセッサを指定する番号」を一定の割合で生成することになる。

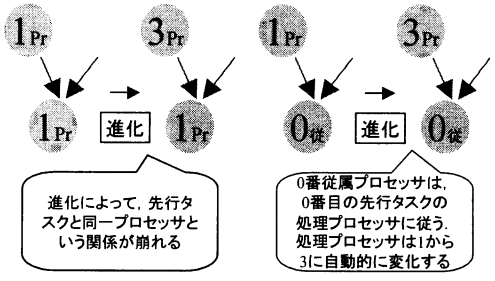


Fig. 8. 従属割り当て法モデル図

5．移住処理

5.1 移住概要

GA処理においては，プロセッサごとにコロニーを生成し，同一コロニー間でのみ交配を行わせ，一定世代ごとに各コロニーにおける最良解を互いに送受信するという手法がよく用いられる。この手法は，初期集団による解の偏り，局所解の収束というGAの問題点の解消，並列性能の効果的な発揮などの面で優れている。過去の研究においても使われてきたが，それらは移住を前提としたGA設計ではなかった。そこで，今回移住を前提にしたGA設計をした。

今回いくつか考えたコーディングにおいて，プロセッサ番号のみGA処理にかけた場合，非常に高速に精度の高い解が得られることが分かった。しかし，プロセッサ番号のみをGA処理する場合はアイドルタスクを考慮するような特殊な解の場合，最適解が絶対得られなくってしまう。一方，従来のプライオリティ遺伝子も使った手法では，速度・精度共に落ちてしまうことになる。

本研究では，プロセッサ番号のみを集中的にGA処理するサブコロニー群を生成し，それらとプライオリティ遺伝子も使った手法のメインコロニーとで移住させる手法をとった。これによりサブコロニー群での荒削りな高速探索と，メインコロニーでの最適解を求め得る確実な探索を目指す。

5.2 移住タイミング

コロニー毎の処理負荷が違うため，世代数で同期を取るのは効率が悪いので，一定タイミングによる非同期での移住を行った。具体的なタイミングとしては，

・サブ⇒サブ　一定世代毎非同期
・サブ⇒メイン　サブコロニーで最良解の更新があった場合。
・メイン⇒サブ　サブコロニーから解の移住を受け取ったとき，その解が最良解以下の場合ならば，メインコロにーでの最良解を送信する

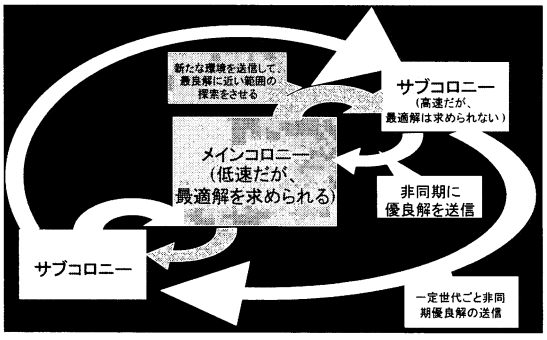


Fig. 9.　メイン･サブコロニー移住モデル図

5.3 移住時の染色体の操作

それぞれの進化してきた環境による染色体の違いを考慮しなければならない。以下はその操作である。

・プロセッサ番号の適合

進化した環境によって，スケジューリングとしては同じ意味であったとしても，遺伝子表現上としてプロセッサ番号が違うという可能性がある。例えば，クリティカルパス上にあるタスクをどのプロセッサで処理しているかなどは，進化した環境次第で変わってしまう。ゆえにこの補正を行う。そのために，プロセッサ番号の使用頻度の度数分布表を作成し，度数順位に従ってプロセッサ番号の変換をする。

・サブコロニーでの遺伝子のタスク順と処理順の並びの相互変換

サブコロニーでは，その処理順に従って遺伝子が並んでいる。移住先の処理順は常に変化している可能性があるので，移住先に送るときは遺伝子の並びをタスク番号順に並び替える操作が必要になる。また移住してきた染色体に関しては自らの処理タスク順表に従い，遺伝子を並び替える。

・処理タスク順表からプライオリティ番号への変換

サブコロニーからメインコロニーへ染色体が移住する際に必要となる操作である。サブコロニーではプライオリティ番号は無視し，処理タスク順表を用いて，それに代替している。従って，メインコロニーへの移住の際には処理タスク順表をもとに，移住染色体のプライオリティ番号を設定する。

・プライオリティ番号から処理タスク順表への変換

メインコロニーからサブコロニーへ染色体が移住する際に必要となる操作である。メインコロニーから移住してきた染色体はプライオリティ番号によって，処理タスク順が決定している。その処理タスク順が，サブコロニーでの処理タスク順よりも良質なものであれば，その変換を行う。

6．GA性能評価

GA性能の評価のための試行を行った。各条件は以下の通りである。

逐次実行環境：x86 Family 15 Model 2 Stepping 7 GenuineIntel 2386 MHz

並列実行環境：SUN Blade 1000 (Ultra SPARCⅢ750MHz ×2)

対象タスクグラフパラメータ

タスク数	：	10, 20, ･･･100
最大直接後続タスク数	：	1～タスク数の一様乱数
最大直接先行タスク数	：	1～タスク数の一様乱数
タスク処理時間（Clock）	：	1～50の一様乱数
通信情報量（Byte）	：	1～500 の一様乱数

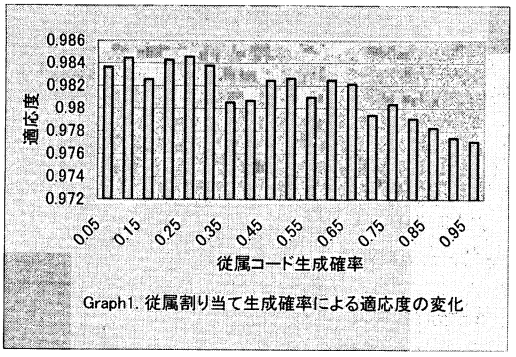
GAパラメータ

GAにおいて使用する各パラメータは過去の実験で得られた最適値を使用している。

交叉率	：0. 75
突然変異率	：0. 05
試行世代数	：4000もしくは2000
通信速度（Bytes／Clocks）	：10
プロセッサ数	：8
移住個体数	：1
移住間隔	：200

6.1 従属割り当て法の性能評価

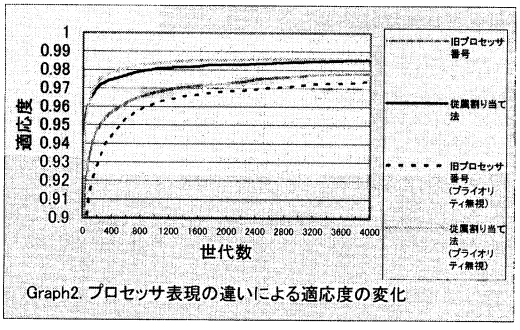
従属割り当て表現を実装するに当たって，通常のプロセッサ番号に対して，従属割り当てコードをどの程度の割合で生成するかを決めなければならない。以下は従属割り当てコードの生成確率の変化による適応度の変化を表した図である。



Graph1. 従属割り当て生成確率による適応度の変化

生成確率が高い場合は性能を下げてしまうことがわかる。以下の試行では統一して，生成確率は0.25とした。

従属割り当て法およびプライリティ番号無視での試行を行った。データは50タスクのスケジューリングを逐次処理で4000世代の試行結果50回の平均値である。



Graph2. プロセッサ表現の違いによる適応度の変化

従属割り当て法の場合，早い世代で解が得られることが分かった。また，プライオリティ番号を無視しても，従属割り当て法の場合，適応度は下がるよりもむしろ上がることが分かる。

プライオリティ番号を無視した場合は実行時間が約半分になった。

Table 1.　逐次実行時間

試行方法	旧プロセッサ番号	従属割当て法	旧プロセッサ番号（プライオリティ無視）	従属割当て法（プライオリティ無視）
時間（sec）	1.785912	1.818667	0.984599	1.055728

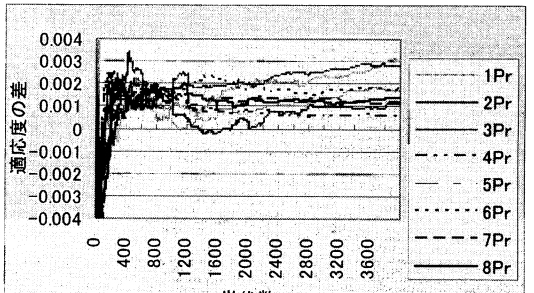
6.2　並列試行の性能評価

並列性能を測るために，以下三通りの方法での試行を行った。

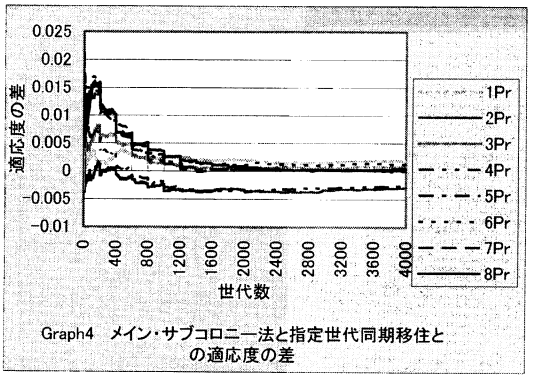
・指定世代同期移住
・指定世代同期移住(プライオリティ番号無視)
・メイン・サブコロニー法

どの方法でも，前述の従属割り当て法を採用している。

データは，タスク数50のタスクグラフを対象にした並列GAスケジューリング50回の平均である。GA試行の世代数は4000回である。並列プロセッサ数は１～8. 各グラフの系列はそれぞれのプロセッサ数を表す。
旧来の指定世代同期移住との各世代の適応度の差を表したグラフをGraph 3およびGraph 4に示す。



Graph3　指定世代同期移住(プライオリティ無視)と，指定世代同期移住との適応度の差



Graph4　メイン・サブコロニー法と指定世代同期移住との適応度の差

プライオリティ番号を無視した場合は，考慮する場合と比べて，おおむね各世代の適応度は向上している。メイン・サブコロニー法は早い世代においては，適応度が向上する場合が多いが，ある程度解が収束してしまうと，それほどの効果は見られない。

6.3　様々なタスクグラフに対する試行結果

汎用的なGA性能を測るために，様々なタスクグラフに対しての試行を行う。タスク数10, 20, ・・・・, 100のタスクグラフをそれぞれ100パターン，合計1000パターンのランダムタスクグラフ対する試行を行った。GA試行は各タスクグラフに対して1回のみ行った。
GA処理の試行世代数は2000世代である。

プライオリティを無視する方法では，4000世代試行す

る。なぜなら，プライオリティ無視法は他の約1/2の実行時間なので，性能測定のためには倍の試行をさせて，実行時間をおおまかに合わせるためである。

メイン・サブコロニー法では，メインコロニーが2000世代に達するまで試行する。サブコロニーはメインコロニーが終了するまで試行を続ける。サブコロニーでの試行世代数はそのプロセッサ性能や対象タスクグラフによって毎回変わることになる。その試行世代数は，おおよそメインコロニーでの試行世代数の1/2〜2倍になった。以下はその試行結果である。各値は，タスク数別の平均適応度，最適解の出現確率を表している。この場合，最適解とはクリティカルパス長と等しいスケジューリング長が得られた場合である。 割り当て可能プロセッサ16での並列GAの実行時間をTable2. に示す。

Table 2. 並列実行時間

試行方法	旧コード 指定世代同期移住	従属割当て法 指定世代同期移住	従属割当て法 (プライオリティ無視)	メインサブ移住
時間(sec)	34.497625	31.329865	29.3068061	28.507524

逐次処理での結果はTable3. のようになった。

Table 3. GA逐次処理の結果

	平均適応度	最適解求解確率(%)
旧コード	0.9848086	6.5
従属割り当て法	0.9915237	24.2
従属割り当て法 (プライオリティ無視)	0.9926012	25.9

従属割り当て法は，非常に高い性能を上げることが分かる。また，プライオリティを無視した試行でも，探索範囲の偏りによる性能低下は見られず，性能が向上している。

次は並列GAでの各種ランダムタスクに対する試行を行った。並列GAでの使用プロセッサ数は４とした。

従属割り当て法は並列型GAにおいても，非常に高い性能を上げることが分かる。また，プライオリティを無視した試行では，かなりの性能向上が見られた。メイン・サブコロニー移住ではそれに比べて性能GA低下してしまった。

次は並列GAでの試行を行う際に，スケジューリングで割り当て可能なプロセッサ数を，従来の8から倍の16に増やしてみた。

Table 4. GA並列処理の結果

	平均適応度	最適解求解確率(%)
旧コード 指定世代同期移住	0.9902157	10.1
従属割り当て法 指定世代同期移住	0.9936519	27.9
従属割り当て法 (プライオリティ無視)	0.99369	29.5
メインサブ移住	0.9936539	29

Table 5. GA並列処理の結果(割当可能プロセッサ数16)

	平均適応度	最適解求解確率(%)
旧コード 指定世代同期移住	0.9879627	7.9
従属割り当て法 指定世代同期移住	0.9942104	28.9
従属割り当て法 (プライオリティ無視)	0.9943553	29
メインサブ移住	0.9943875	29.7

割り当て可能なプロセッサが増えれば，探索範囲は増えるが，スケジューリングの解は向上する可能性がある。旧来のコーディングでは，探索範囲の増加のせいで解精度はかなり低下していたが，従属割り当て法では，解の精度はさらに向上している。また，プライオリティを無視した場合の方が解精度がさらに向上しているが，最適解の求解確率が落ちてしまった。メイン・サブコロニー法では適応度，最適解の求解確率ともに向上している。

7. 考 察

並列実行では，プライオリティを無視した場合では，1世代あたりの実行時間が約半分ほどになる。この点を考えれば，メイン・サブコロニー法での性能向上は，プライリティ無視手法での性能との比較からも，単にサブコロニー(プライオリティ無視)での試行世代数が通常よりも増加するためと考えられる。しかしながら割り当てプロセッサを増やした場合は，最も性能が良くなっているので，移住による相互作用もある程度発揮されていると見ることができる。

8．おわりに

先行タスクと同一のプロセッサで処理するという意味をプロセッサ遺伝子に追加した「従属割り当て法」は，逐次・並列ともに対象タスクグラフによらず高性能を示した。

タスクの処理順を決めつけ，割り当てタスクのみを処理するという，プライオリティ無視手法では，処理時間を半分ほどにでき，解精度もおおむね向上するという利点があることが確認された。しかし，探索範囲に最良解があるとは限らないという欠点がある。

その欠点を改良すべく「メイン・サブコロニー法」を考案した。従来の指定世代同期移住法と比較して，より良い解精度を得た。しかし，サブコロニーでの試行世代数の増加による性能向上が強いと思われ，サブコロニーでの探索範囲の変化など，移住による相互作用による性能向上は思ったほど発揮できていないと思われる。

総評として，従来のGA手法に比べれば，短時間で実行でき，適応度もかなり高くなり，最適解を得られるケースが大分多くなったと言える。

参考文献

北野宏明：「遺伝的アルゴリズム２」，産業図書，1995
儘田勲夫：「通信を考慮したタスクスケジューリング問題のGA並列解法」，平成11年度成蹊大学ソフトウェア工学研究室卒業論文