遺伝的アルゴリズムを用いた スレッドの協調実験

原 章　　佐藤 泰介

東京工業大学 情報理工学研究科 計算工学専攻
東京都目黒区大岡山 2-12-1
03-5734-2186
ahara@cs.titech.ac.jp　　sato@cs.titech.ac.jp

あらまし
スレッドはプロセス中の制御の流れであり、１つのプロセス中に複数のスレッドを走らせるマルチスレッドプログラムでは並行処理が可能となり処理の高速化などに利用できる。マルチスレッドプログラムを効率的に動作させるには、セマフォなどを用いてスレッドの同期を実現する必要があるが、同期の問題はタイミングに依存するので設計が困難である場合が多い。
そこで本論文ではセマフォの操作を表す列に対して遺伝操作を行ない複数のスレッドが協調作業を行なうように進化させるシステムを提案する。このシステムを「食事をする哲学者の問題」に対して適用し実験を行なった結果、全体で協調作業を行なうように各スレッドの行動が進化することを確認した。
キーワード　　遺伝的アルゴリズム、 スレッド、 食事をする哲学者の問題、 セマフォ

Experiments on the cooperation of threads with genetic algorithms

Akira Hara　　Taisuke Sato

Department of Computer Science, Graduate School of Information Science and Engineering,
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo, 152, Japan
03-5734-2186
ahara@cs.titech.ac.jp　　sato@cs.titech.ac.jp

Abstruct
Threads are flows of control within a process. The multithreaded program, having multiple threads of control within a process, enables parallelism, and we can use it for improving performance. In order for a multithreaded program to work efficiently, it is necessary to realize synchronization of threads by semaphores. It is however difficult to design semaphores, which depend on the thread scheduling.
In this paper, we propose technique that makes threads evolve their actions to cooperate by performing genetic operations on chromosomes, each representing semaphore operations. We applied this technique to "dining philosophers problem" and observed that threads evolved to cooperate one another.

Keywords　　genetic alogorithms, thread, dining philosophers problem, semaphore

1　はじめに

遺伝的アルゴリズムは生物の進化の過程から着想された確率的な探索手法である [1, 2, 3]。これは与えられた問題の解を表す個体を複数個ランダムに発生させ、個体に対して交叉や突然変異などの操作を用いて適者生存の原理に基づく世代交代を行ない与えられた問題に適した個体を作り出そうとするものである。これは組合せ問題や探索問題に対して実用上の最適解を速やかに得ることができる有効な手法である。また遺伝的アルゴリズムは人工生命などの進化・適応システムの構築のための基本的なモデルとも考えられる。

一方、スレッドはプロセス中の制御の流れであり、1 つのプロセス中に複数のスレッドを走らせるマルチスレッドプログラムでは並行処理が可能となり処理の高速化などに利用できる。

マルチスレッドプログラムを効率的に動作させるにはスレッドの同期を実現する必要があるが、同期の問題はタイミングに依存するので設計、診断が困難である場合が多い。

そこで本論文では遺伝的アルゴリズムによりスレッドの行動を進化させ、複数のスレッドの協調を実現することを目標とし、並行プログラミングの同期の典型的な問題である「食事をする哲学者の問題」に対してシミュレーションを行なった。この問題は複数のスレッドが共有資源の要求を行なうもので、デッドロックを引き起こさず効率良く動作するにはスレッド間の協調が必要となるものである。

本論文では遺伝的アルゴリズムによりスレッドの同期を実現するシステムを考案し、これを適用した「食事をする哲学者の問題」に関する実験システムの構築、実験、その結果に対する考察を行う。

2　スレッド (thread)

UNIX などの計算機の中に多数のプロセスが走っているように、ユーザが一つのプロセスの中に多数の「軽量プロセス」(lwp: light weight process) を作りタイムシェアリングシステムのように走らせることができる。これにより一つのプロセスの中で複数の処理を並行に実行することができる。この「軽量プロセス」のことをスレッドと呼ぶ [4, 5, 6]。

図 1のように計算機に対するプロセスの関係と

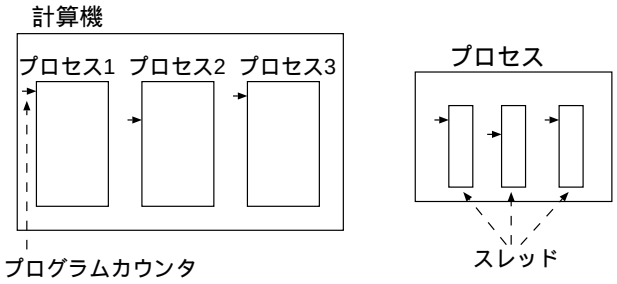

図 1: プロセスとスレッドの関係

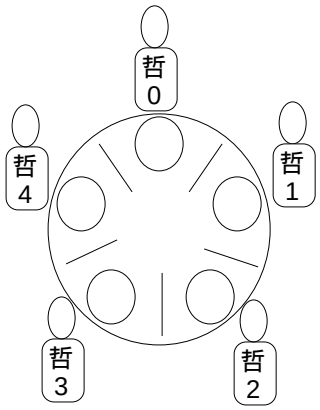

図 2: 食事をする哲学者の問題

プロセスに対するスレッドの関係は同じようにみなすことができる。ただし 1 つのプロセス内の異なるスレッドは異なるプロセスとは違って完全に独立しているわけではない。スレッドはアドレス空間、広域変数、オープンしているファイル、シグナル、セマフォなどを共有している。

プログラムをマルチスレッド化することの利点には次のようなものがある。

- 並列性による計算速度の向上
- プログラムの設計、コーディングの単純化
- システムリソースの効率利用

また 1 つのプロセス内の複数のスレッドは同一の広域変数やオープンしているファイルなどを共有するので、スレッド間の通信が行ない易くスレッド同士が協調して作業を行なうのに適している。

3　食事をする哲学者の問題

3.1　食事をする哲学者の問題

1965 年に Dijkstra によって提案された同期化問題が「食事をする哲学者の問題」である [6, 7, 8]。問題は次のようなものである。

5 人の哲学者がテーブルのまわりに座っている。各哲学者の前には 1 つのご飯のお皿があり、その皿の左右には箸が 1 本ずつ置かれている。ただし、皿と皿の間には箸は 1 本しか置かれていない。(図 2参照)

哲学者は、食事には 2 本の箸を必要とし、それぞれ思索と食事を交互に繰り返す。

このときどのようにすれば哲学者達は効率的に思索と食事を実行できるだろうか。

解決策

この問題では、哲学者が並行プロセス (スレッド) に相当し、箸が共有資源に相当する。哲学者は 2 本の箸を必要とするので、隣り合った哲学者は同時に食事することはできず、5 人のうち同時に食べられるのは 2 人までである。

セマフォを用いて 2 人の哲学者が同じ箸を同時に使うことがないようにするための解決策として次のようなものが考えられる。

- 各哲学者は次のような繰り返しの動作を行なう。

```
for(;;){
    think;
    take_left_chopstick;
    take_right_chopstick;
    eat;
    put_left_chopstick;
    put_right_chopstick;
}
```

- それぞれの箸に対してセマフォを 1 つずつ用意し、初期値を 1 とする。

- 哲学者は箸を取るときにそのセマフォを 1 つ down し、箸を置くときにそのセマフォを 1 つ up する。

この方法により 2 人の哲学者が 1 本の箸を同時に使ってしまうことは避けることができる。しかし 5 人の哲学者が同時に自分の左側の箸を取ってしまった場合、全員が右の箸を永久に待ち続けるデッドロックと呼ばれる状況が発生する。よってこれは解決策として不十分である。

4　遺伝的アルゴリズムを適用した「食事をする哲学者の問題」

4.1　哲学者の行動

前節で示した解決策では箸への相互排除は実現できたが、哲学者の動作のタイミングによってデッドロックが発生するといった問題があった。

そこで各哲学者の同期を取るために、次のようなセマフォを導入して前節の解決策を拡張する。

- 各哲学者に対して、お互いの動きを制御するためのセマフォを用意する。

- 哲学者は、左の箸を取る行動に移る前に、自分に対して値 1 の down 操作を行ない、食事が終わり 2 本の箸を置いた後、指定された哲学者に対する up 操作を行なう。

よって各哲学者の行動はつぎのようになる。

```
for(;;){
    think;
    自分のセマフォに対する値 1 の down 操作;
    take_left_chopstick;
    take_right_chopstick;
    eat;
    put_left_chopstick;
    put_right_chopstick;
    指定された哲学者のセマフォへの up 操作;
}
```

自分の持つセマフォの値が 0 の場合は、自分のセマフォに対する down 操作が実行できず sleep 状態となり他の哲学者からの up 操作を待つことになる。

この同期用セマフォの初期値と周りの哲学者に対する up 操作の値を染色体上に表し遺伝的アル

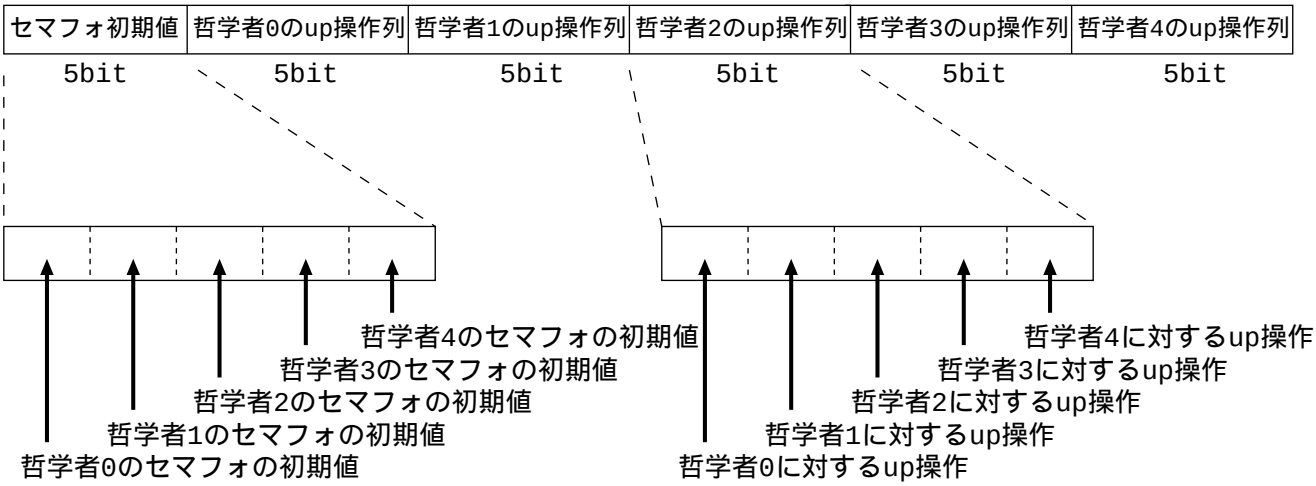

図 3: 染色体のコーディング

ゴリズムにより行動を進化させることによって、デッドロックが発生せず、効率良く食事ができるような哲学者の同期を実現させる。

4.2 染色体のコーディング

同期用のセマフォの初期値と 5 人の哲学者が行なう up 操作の列を 1 つの染色体に組み込む。染色体は固定長の文字列で 0 と 1 の 2 値で表す。染色体は図 3のようになり、セマフォ初期値の列、哲学者の up 操作の列ともに左側から順に哲学者 0 から 4 に対応する。up 操作を表すビットが 1 ならば値 1 の up 操作を行ない、0 ならば up 操作を行なわない。

例えば次のような染色体があるとする。

10010 01010 00001 01101 00000 10100

この染色体は次のことを表す。

- 哲学者 0 と哲学者 3 のセマフォの初期値を 1 とし、哲学者 1、哲学者 2、哲学者 4 のセマフォの初期値を 0 とする。
- 哲学者 0 は、哲学者 1 と 3 に対して up 操作を行なう。
- 哲学者 1 は、哲学者 4 に対して up 操作を行なう。
- 哲学者 2 は、哲学者 1 と 2 と 4 に対して up 操作を行なう。
- 哲学者 3 は、どの哲学者に対しても up 操作を行なわない。
- 哲学者 4 は、哲学者 0 と 2 に対して up 操作を行なう。

4.3 各染色体の適応度の計算方法

各個体ごとに「食事をする哲学者の問題」を実行して、その結果から各染色体の適応度を計算する。

「食事をする哲学者の問題」の解決策に要求されるものを次のように設定する。

- 無矛盾性 (1 つの箸を 2 人の哲学者が同時に使用することはない。)
- 安全性 (デッドロックを起こさない。)
- 効率性 (一定時間内にできるだけ多く食事ができた方が良い。)
- 公平性 (各哲学者が食事をする回数は等しい方が良い。)
- 資源の有効利用 (左の箸を持ったまま、右の箸を待つことは資源の無駄である。)

無矛盾性は箸に対してセマフォを導入したことで解決されているので、残りの 4 つを遺伝的アルゴリズムにより実現する。また哲学者をセマフォにより制御し箸に対する同時アクセスを無くすため、左の箸を取ろうとして待たされるという動作も無駄であると考える。

そこで作成したプログラムでは、1 つの染色体に対して一定時間「食事をする哲学者の問題」を走らせ、その間の

- 食事ができた延べ人数 $(total)$
- 各哲学者の食事回数
- 左の箸を持ったまま、右の箸を待った回数 $(rightwait)$
- 左の箸を待った回数 $(leftwait)$

を数える。また哲学者全員が自分に対する down 操作ができない状況が起これば、$deadlock$ を 1、起こらなければ 0 とし、各哲学者の食事回数から分散 $(variance)$ を計算する。

これらの要素を $Efficiency,\ UnfairPenalty,\ RightPenalty,\ LeftPenalty, DeadLockPenalty$ というパラメータで重み付けし、各染色体の適応度を次のように計算する。

$$\begin{aligned} fitness \;=\; & Efficiency \times total \\ & + UnfairPenalty \times variance \\ & + RightPenalty \times \frac{rightwait}{total} \\ & + LeftPenalty \times \frac{leftwait}{total} \\ & + DeadLockPenalty \times deadlock \end{aligned}$$

プログラムは各哲学者の行動をスレッド化して並行に動作させる。

またどの哲学者も食事時間、思索時間は一定とする。

5 実験・考察

5.1 遺伝的アルゴリズムを適用した「食事をする哲学者の問題」

前章で説明したシステムを用いて「食事をする哲学者の問題」の実験を行なった。遺伝的アルゴリズムのパラメータは、表 1のように設定した。

この実験での世代数と各個体の適応度の中の最大値の関係を表すグラフを図 4に、世代数と各世代の箸待ち回数の平均を表すグラフを図 5に示す。

表 1: 遺伝的アルゴリズムのパラメータ

各世代の個体数	20
選択	トーナメント方式、 トーナメントサイズは 5
交叉	一様交叉、交叉率は 0.8
突然変異率	0.033
$Efficiency$	0.05
$UnfairPenalty$	-0.8
$RightPenalty$	-2.0
$LeftPenalty$	-2.0
$DeadLockPenalty$	-20

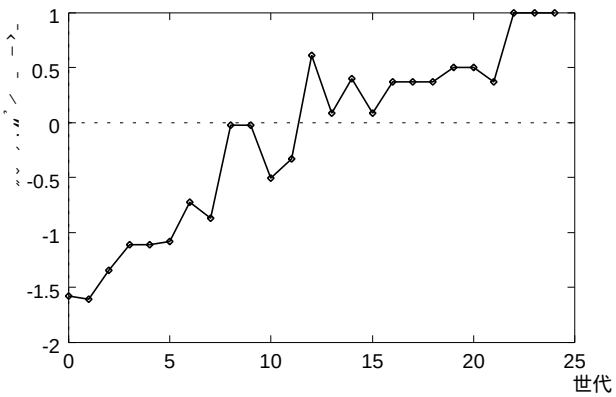

図 4: 世代数と適応度の最大値の関係

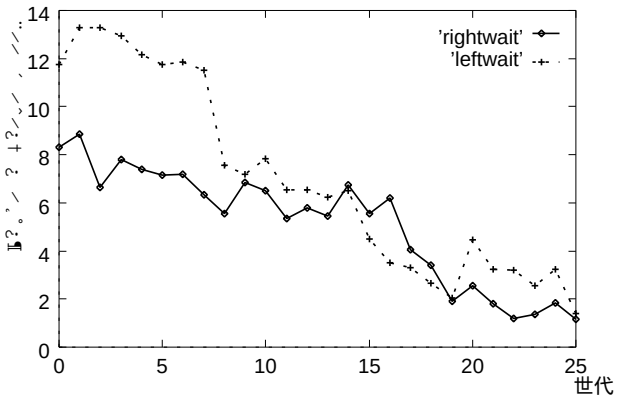

図 5: 各世代の箸待ち回数の平均

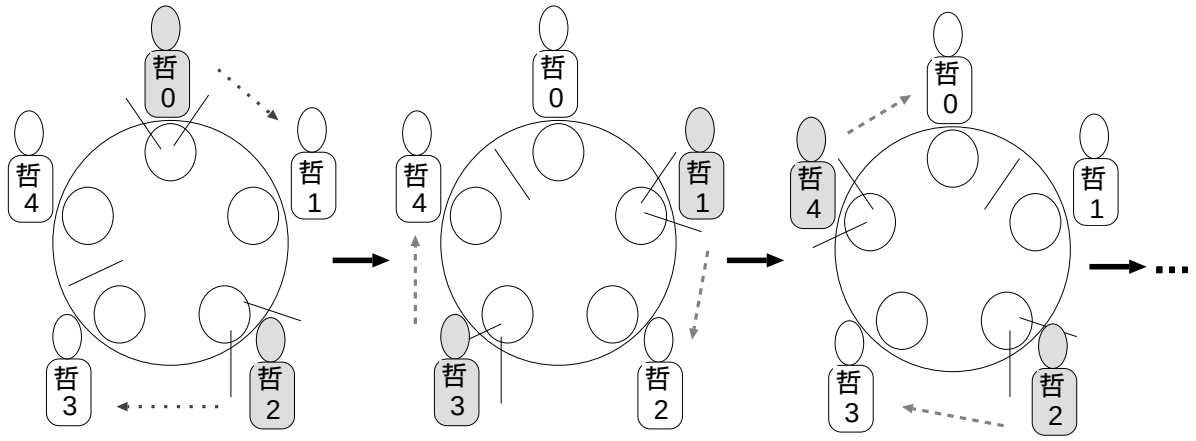

図 6: 適応度の最も高い染色体による哲学者の食事の様子

これらのグラフから、世代数が進むにつれて哲学者の左右の箸を待つ回数が減少し、適応度の高い個体が出現していることがわかる。

実行終了後の適応度の最も高い染色体として、次のようなものが得られた。

10100 01000 00100 00010 00001 10000

この染色体は、哲学者 0 と 2 にセマフォの初期値として 1 を、その他の哲学者には 0 を与え、各哲学者は自分の次の番号の哲学者 (ただし哲学者 4 は哲学者 0) に対して up 操作を行なうことを表す。すなわちこの染色体は

- 哲学者 0 と哲学者 2 から食事を始める。
- 各哲学者は食事後、自分の左隣の哲学者を起こす。

ことを意味している。

各哲学者がこの行動を取ることにより、常に 5 人の哲学者の内 2 人が食事ができ、食事をしている哲学者以外は起こされるのを待っているので箸に対する競合も無い。また同時に食事をしようとする哲学者は高々2 人なのでデッドロックは発生しない。各哲学者の食事回数も公平である。これは設定した要求を満たした解になっている。このときの全体の食事の様子を図 6に示す。

その他にある程度適応度の高いものとして, 次のようなものが得られた。

染色体 **a** 00010 00010 10000 01000 00101 00000

染色体 **b** 01000 01000 00101 00010 00000 10000

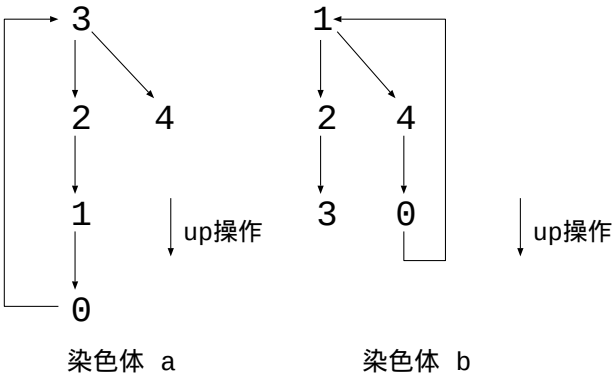

図 7: 適応度の高い染色体による哲学者の食事の順序

各々の染色体による実行の様子を図 7に示す。この図は哲学者の食事の順序を示すもので、図中の数字は哲学者の番号を表し、それらが同じ行にある場合は同時に食事をすることを表す。これらは最も適応度の高い染色体に比べると常に 2 人が食事ができるわけではなく並列性は劣るが、同時に 5 人が食事をすることはないのでデッドロックが発生せず、各哲学者の食事回数も公平であり設定した要求を満たした解になっている。

5.2 哲学者の人数を 10 人に変えての実験

「食事をする哲学者の問題」の哲学者の人数を 10 人に変更して、遺伝的アルゴリズムを適用した。遺伝的アルゴリズムのパラメータは、表 2のように設定した。

表 2: 哲学者の人数を 10 人にした場合のパラメータ

各世代の個体数	20
選択	トーナメント方式、 トーナメントサイズは 5
交叉	一様交叉、交叉率は 0.8
突然変異率	0.0091
$Efficiency$	0.02
$UnfairPenalty$	-0.8
$RightPenalty$	-2.0
$LeftPenalty$	-2.0
$DeadLockPenalty$	-20

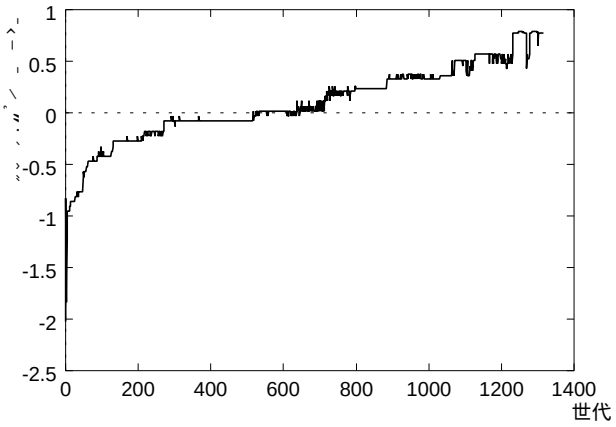

図 8: 哲学者が 10 人の場合の世代数と適応度の最大値の関係

この実験での世代数と各個体の適応度の中の最大値の関係を表すグラフを図 8に示す。世代数が進むにつれて、適応度の高い個体が出現していることがわかる。

実行終了後の適応度の最も高い染色体として、次のようなものが得られた。

0100010010 0100000000 0000000000 0000000000 0000000010 0001010101 1010101000 0000000000 0000000000 0000000000 0000000000

この染色体を用いて「食事をする哲学者の問題」を動作させた時の実行の様子を図 9に示す。哲学者が 10 人の場合、同時に食事ができるのは最大で 5 人である。得られた解は全員が同時に食事をしようとすることはないのでデッドロックは発生しない。また奇数番号の哲学者と偶数番号の哲学者

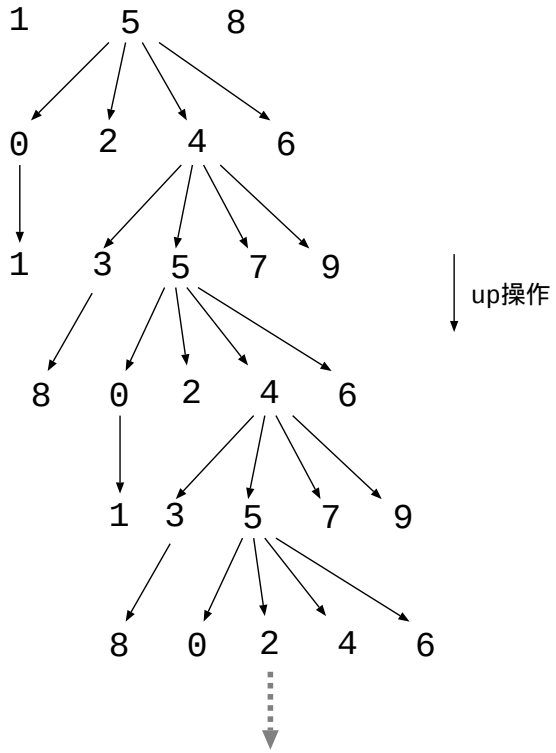

図 9: 適応度の高い染色体による 10 人の哲学者の食事の順序

が交互に食事をする効率の良い動作が実現されており、各哲学者の食事回数も公平である。これは設定した要求を満たした解になっている。

5.3 問題点

この実験の問題点を以下に挙げる。

- 得られた解は各染色体を一定時間動作させた結果により適応度を評価したものである。その時間内にデッドロックが発生せず、公平性が得られたからといって必ずしも要求を満たすとは限らない。要求を満たすかどうか得られた解の解析が必要である。

- 前節で要求を満たす解が得られたが、この解は全ての哲学者の食事に要する時間と思索に要する時間は常に一定で等しいという場合に限られたものである。哲学者ごとに食事や思索の時間が違う場合は、この解では食事の効率が悪くなることが考えられる。

- 哲学者の人数が 5 人の時は有効な解が得られるが、哲学者の人数を 10 人に増やすと適応度の上昇が止まり解が得られないことが非常に多くなり、それ以上の人数では有効な解が得られなかった。これは哲学者の不要な up 操作が多く、交叉や突然変異によってその up 操作が減少しても外から観測できる食事回数や箸を待つ行動に変化が現れず個体の適応度に反映されないため、哲学者の行動が進化しないと考えられる。

6 まとめ

本論文の目的は遺伝的アルゴリズムによるスレッドの協調の実現であった。そのために次のようなシステムを作成した。

- 複数のスレッドを並行に動作させる。
- 各スレッドはセマフォを持ち、自分や他のスレッドに対してセマフォ操作を行なう。
- セマフォ操作の値や対象を染色体として表し、遺伝的アルゴリズムを適用する。

このシステムを「食事をする哲学者の問題」に適用すると、全体の性能が向上するように各哲学者の行動が進化し協調動作を行なうことを確認した。

今回作成したシステムではセマフォ操作の値が限定された簡単なものであったが、遺伝的アルゴリズムを適用することにより、「食事をする哲学者の問題」の効率的な動作を実現できた。これは大規模で複雑な並行処理における協調を実現するための手がかりになったと考えられる。

今後の課題としては、セマフォ操作が様々な値を取り、周囲の状況に応じた行動を選択するようにシステムを改良し、より複雑な同期を実現する必要がある。また、より多くのスレッドでの同期を実現する必要もある。

参考文献

[1] 伊庭斉志:“遺伝的アルゴリズムの基礎”, p.59–104, オーム社 (1994)

[2] 安居院猛, 長尾智晴:“ジェネティックアルゴリズム”, p.1–27, 昭晃堂 (1993)

[3] 北野宏明編:“遺伝的アルゴリズム”, p.1–41, 産業図書 (1993)

[4] Steve Kleiman, Devang Shah, Bart Smaalders :“Programming with THREADS”, p.1–12, SunSoftPress/Prentice Hall PTR(1996)

[5] Bil Lewis, Daniel J.Berg・著, 岩本信一・訳:“マルチスレッドプログラミング入門”, p.29–50, アスキー (1996)

[6] A.S.Tanenbaum・著, 引地信之, 引地美恵子・訳:“OS の基礎と応用”, p.33–84,573–591, トッパン (1995)

[7] M.Ben-Ari・著, 渡辺栄一・訳:“並行プログラミングの原理” ,p.137–148, 啓学出版 (1986)

[8] 重松保弘:“UNIX プログラミング入門講座”, p.117–146, アスキー (1994)