

再利用を用いた GA の高速化

鈴木 郁真[†] 池内 康樹[†] 津 邑 公 暁[†]
中 島 康 彦^{††,†††} 中 島 浩[†]

遺伝的アルゴリズムにおいて最も処理時間を要する適合度計算に対し、再利用を適用することで高速化する手法を提案する。適合度計算の入力となる遺伝子が、前世代で処理された遺伝子と多くの共通部分を持つことから、適合度関数を分割することで再利用の効果を引き出す手法について述べる。GENEsYs を用いて評価した結果、平均 29%、最大 83% のサイクル数を削減できた。更に、関数分割などの改良を施すことにより、平均 38%、最大 87% までこれが向上した。特に適合度計算に要する時間が長い適合度関数について、再利用の効果が非常に大きくなることが分かった。

Fast GA with Reuse

IKUMA SUZUKI,[†] YASUKI IKEUCHI,[†] TOMOAKI TSUMURA,[†]
YASUHIKO NAKASHIMA^{††,†††} and HIROSHI NAKASHIMA[†]

This paper describes a speedup technique with computational reuse for the fitness calculation of GA programs. A genotype has many genes in common with its parental genotypes. Therefore, partial results of fitness calculation are reusable. Through the result of an evaluation with GENEsYs, a well-known GA software, we show that the maximum ratio of the cycle reduction reaches 83%, while accomplishing average reduction of 29%. Furthermore, dividing fitness procedures raises the maximum ratio to 87% and average ratio to 38%.

1. はじめに

遺伝的アルゴリズム (Genetic Algorithm: 以下 GA) は、解探索に古くから用いられている有効な手法のひとつである。しかし、GA はその有効性ゆえに解決困難な問題に適用されることがほとんどであり、現在の計算機性能をもってしても、膨大な処理時間を要する場合が多い。

このため、現実的な時間で有効な解を得るためには、並列化を用いた GA の高速化が不可欠とされてきており、理論と実装の両面において多くの並列 GA の研究が行われている¹⁾。一般に並列 GA では、プロセッサ間で個体集団の移送を行う必要があり、これが高速化する際のボトルネックとなるため、個体集団をコンパクトに表現することで移送コストを軽減する研究なども行われている²⁾。

そこで本稿では、再利用³⁾を用いた、並列化とは全く着眼点の異なる GA の高速化手法を提案する。再

利用とは、関数などの命令区間の実行時にその入出力の組を記憶し、再び同じ入力によってその命令区間が実行されようとした場合、記憶してある出力を書き戻すことで命令区間の実行自体を省略する高速化手法である。

GA の処理において、前世代と現世代の個体の遺伝子表現に共通部分が多いことから、個体の適合度計算などの部分に再利用が効果的に適用可能であることを示す。ただし遺伝子表現は過去に出現した遺伝子表現と完全には一致しないことから、再利用を効果的に適用するためのプログラミング手法について述べ、評価結果によりその効果を示す。再利用による GA 高速化は、従来の並列 GA と独立に適用可能な手法である。よってこれらを併用することで、更なる高速化も可能となると考えられる。

以下 2 章では、適用手法である再利用の概要について述べ、3 章で、GA の一般的な処理プロセスについて概説しながら、再利用の適用可能性について探る。4 章では具体的な再利用の適用手法について述べ、5 章で汎用 GA ソフトウェアである GENEsYs を用いた評価結果を示す。

[†] 豊橋技術科学大学
Toyohashi University of Technology

^{††} 京都大学
Kyoto University

^{†††} 科学技術振興機構さきがけ研究 21
PRESTO, JST

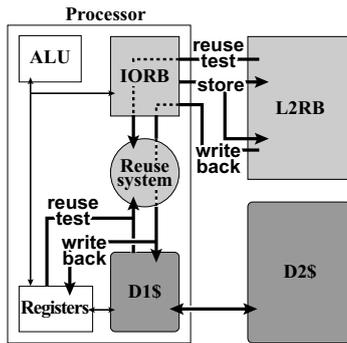


図 1 再利用機構

2. 再利用

本章では、GA の高速化手法として提案する再利用の概要について述べる。

2.1 概要

再利用は、プログラムを構成する命令区間を多入力・多出力の複合命令としてとらえ、過去に行った複合命令の実行結果を記録しておくことで、同一入力による当該複合命令の実行自体を省略する高速化手法である。従来多くの研究が行われてきた値予測および投機的実行は、数多くの命令の投機あるいは実行結果の破棄が必要であるのに対し、再利用は実行する必要のある命令列そのものを削減できるという点で、従来とは発想の異なる高速化技術である。

我々の実行モデルでは、再利用対象とする命令区間として、多くの命令を含みかつ始点と終点を容易に特定できる、関数およびループを仮定する。これにより、再コンパイルや、静的解析に基づく付加情報埋め込み(バイナリアノテーション)を必要とせず、既存バイナリに変更を加えることなく高速化が可能となる。

2.2 入出力記録表

命令区間にとって入力とは、レジスタおよび主記憶に対する参照である。命令区間は、レジスタおよび主記憶を参照することで入力を得、処理を行った結果をレジスタおよび主記憶に書き戻すことで出力を行う。このとき、入力値が等しい間は出力も同じであり、入力が異なった命令以降の出力は枝分立的に派生していく。つまり参照順に入力を並べた場合、レジスタ番号や主記憶アドレスをノードとし、その格納値を枝とするような多分木に含まれる 1 パスとして、ある入力セットは表現される。

再利用を実現するためには、このような多分木で表現された、過去に実行された命令区間の入力セットを記憶するための再利用表 (reuse buffer) が必要となる。さてこの再利用表は、命令区間実行時には単に入力の一致検索が行われるだけでなく、命令区間による出力の書き込みや、自らが書き込んだ出力の再度読

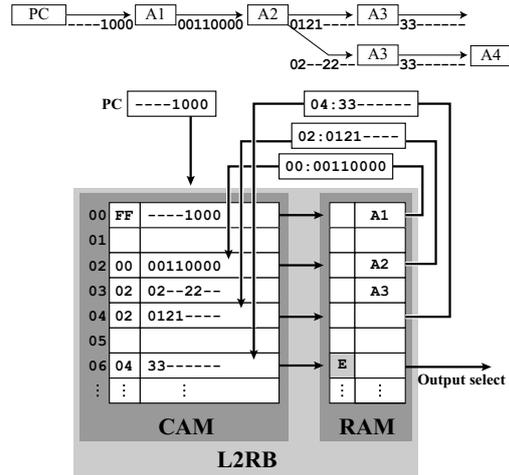


図 2 L2RB の構造と動作

み出しなど、頻繁に入出力処理を行う必要がある。多分木構造は、多種の入力パターンを表現するには適しているものの、このような頻繁な読み書きが行われる場合はその低速さが問題となってしまう。よって、単一の入力パターンを記録できる小規模かつ高速な再利用表 (以下、IORB) と、過去の出現した複数の入力パターンを格納する再利用表 (以下、L2RB) を用意する。命令区間実行中は IORB を用い、命令区間実行終了時に IORB から L2RB に入力セットを保存することにより、効率のよい再利用表を実現する。

2.3 再利用機構

我々の実行モデルが想定する再利用機構の概要を図 1 に示す。プロセッサは、命令区間実行中に L2RB を参照し (reuse test)、入力の一致比較を行うと同時に、IORB に入出力セットを登録しながら命令区間を実行する。入力セットが完全に一致するエントリが見つかった場合、当該エントリに対応する出力を L2RB からキャッシュおよびレジスタに書き戻す (write back)。これにより命令区間の実行が省略される。入力が一致しなかった場合、命令区間実行終了時に IORB の内容を L2RB に格納する (store)。

ここで再利用表、特に L2RB における入力一致比較のための連想検索コストが、再利用に要するオーバーヘッドの大部分を占める。よって L2RB は、連想検索を高速に行える必要がある。本機構ではこの L2RB を中容量の汎用 CAM (Content-Addressable Memory) を用いて構成することを想定しており、これにより再利用オーバーヘッドを比較的小さく抑えることができる。

L2RB の構成を図 2 に示す。前述のように、入力パターンは図 2 上部に示したような多分木で表現できる。これを、L2RB 内の CAM 部に枝、RAM 部にノードを対応させることで、折畳んで格納する。CAM 部の各エントリには親エントリを表すインデックスを用意し、入力値と組み合わせたものをキーとして検索を進めて

いく。アドレスを格納する RAM 部のエントリには終端フラグを設けることで、入力一致比較の終端を示すとともに、可変長の入力セットを格納可能とする。

3. 遺伝的アルゴリズム

3.1 概要

GA は、生物の進化プロセスをもとにして構築された解探索手法である。定式化することが難しいと考えられる問題に対し、進化を計算機上でシミュレートすることで、有効に解の探索を行うことを目的として使用される。

具体的には、対象となる問題の解を遺伝子表現 (genotype) にコーディングし、さまざまな遺伝子表現で表される個体同士で生殖活動を行わせることで次世代の子孫を作りだしながら、それらの個体の適合度に応じて選択・淘汰を行っていくことで、集団全体の適合度を向上させ、これを繰り返すことで最適な解を探索する。

3.2 プロセス

GA は一般的に、生殖、適合度計算、個体選択というプロセスで 1 世代の処理が構成され、これを繰り返し行うことにより解探索を行う。本節では、GA に再利用を適用するに先立ち、これらの処理プロセスを順に概説し、それぞれの再利用適用可能性について考察する。

生殖

生殖処理では、遺伝子表現に対して交叉 (crossover) および突然変異 (mutation) の処理を行い、次世代の遺伝子表現を生成する。

交叉は、まず一定の確率 (交叉率) に従い個体を選択し、選択された個体同士で一部の遺伝子 (gene、遺伝子表現の構成要素) を交換する操作である。交叉手法には、遺伝子表現内の一点でのみ交叉を行う一点交叉、多点で行う n 点交叉、全ての遺伝子を両親の同位置の遺伝子のどちらかからランダムに選択する一様交叉などがある。 n 点交叉 ($n = 2$) の例を図 3 に示す。

突然変異は一定の確率 (突然変異率) に従って、全個体のなかから遺伝子が選択され、その遺伝子に対して書き換えを行う操作である (図 4)。遺伝子が単純な 0/1 で表現されている場合は、当該遺伝子に対してビット反転が行われる。

さて、生殖における主な処理は、遺伝子表現の変更であり、プログラム中ではそのほとんどは配列のコピー処理で占められる。これにおける計算量は軽微であり、再利用の適用可能性を考えた場合、再利用オーバーヘッドなども考慮すると、効果は低いと考えられる。

適合度計算

次に、生殖で生成された各個体の適合度の計算を行う。一般には、遺伝子表現を入力とし適合度を計算・出力する関数を用意し、その関数を適用することで各

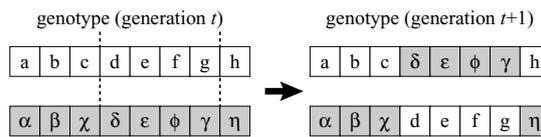


図 3 n 点交叉 ($n = 2$)



図 4 突然変異

個体の適合度を算出する。この際、遺伝子表現は関数の入力として扱いにくいいため、一般に遺伝子表現に対して何らかの変換が行われたうえで、適合度が計算される。なお、生殖の対象とならず全世代から変化していない遺伝子表現に関しては、一般に適合度計算は省略される。

適合度計算は、GA において最も時間を要する処理であり、この部分を高速化することによって GA 全体の所要時間を大きく削減できる可能性がある。さて、適合度計算関数の入力となる遺伝子表現は、前世代に存在した遺伝子表現の交叉によって生成されたものであり、その一部は常に前世代の個体と共通している。つまり、適合度計算において構成遺伝子の部分集合に対する処理が含まれている場合は、その処理は再利用が可能であると考えられる。また、要する処理量から考えても、適合度計算は再利用に適した処理であると言える。

個体選択

算出された適合度の高いものほど多産であり、低いものほど淘汰されやすくなるように個体を選択する処理である。適合度に比例した割合で個体を次世代に残す方法、適合度の高いものだけを残す方法、集団からランダムに個体集団の部分集合を選択し、その中で最も良い個体を選択することを繰り返すことで次世代の個体集団を生成する方法などがある。

個体の選択処理では、すでに計算された適合度に基づく比較やソートがその多くを占めるため、やはり生殖と同様処理量が軽微であり、再利用適用による効果は低いと考えられる。

4. 再利用の適用

前章で述べたように、最も再利用の効果が目立つのは適合度計算である。しかし、処理対象となる遺伝子表現は過去に出現した遺伝子表現と完全には一致していないため、再利用の高い効果を得るためにはプログラミングに工夫が必要である。本章では、適合度の算出プロセスにおいて、再利用率を向上させる方法について述べる。

分割前	分割後
<pre>double f(g, l) double g[]; int l; { int i; double sum = 0.0; for(i=0; i<l; i++){ sum += g[i]*g[i]; } return(sum); }</pre>	<pre>double f(g, l){ double g[]; int l; { int i; double sum = 0.0; for(i=0; i<l; i+=6){ sum += f2(g[i], g[i+1], g[i+2], g[i+3], g[i+4], g[i+5]); } return(sum); } double f2(g1, g2, g3, g4, g5, g6) double g1, g2, g3, g4, g5, g6; { return(g1*g1 + g2*g2 + g3*g3 + g4*g4 + g5*g5 + g6*g6); } }</pre>

図 5 適合度関数の分割例

4.1 遺伝子の格納アドレス

前述のように処理対象となる遺伝子表現は前世代からその一部を受け継いでいるため、入力セットのサブセットは過去に用いられた入力セットと一致し、再利用による効果が見込める。

しかし、一般に個体が異なる遺伝子表現はプログラム上で別配列に格納されており、実行時の主記憶値アドレスが異なる。すなわち、適合度計算の入力値が一致する場合でも、その値が格納されているアドレスが異なるため、再利用機構は同一入力であるという判定を行うことができない。

これを回避するには、1つの遺伝子表現が格納できる配列を用意し、処理対象となる遺伝子表現を常に一度その配列にバッファリングした上で適合度計算を行うようにする。これにより、入力アドレスの違いが解消でき、再利用率を十分に引きだすことが可能となる。

4.2 入力遺伝子数

適合度計算の入力には、当然ながら遺伝子表現に含まれる全ての遺伝子が使われる。しかし遺伝子表現は、過去に出現した遺伝子表現とは部分的には共通しているものの完全に同じではない。つまり、遺伝子表現全体を処理対象とする適合度計算関数は、入力値が過去と完全に一致することはなく、再利用の効果が望めない。

ただし一般に適合度計算は、遺伝子表現の一部のみ依存する処理に分割できる可能性が高い。このような場合、遺伝子表現の一部を処理するサブ関数を定義し、それらの結果から適合度を計算するようにすることで、そのサブ関数に対する再利用の効果が望める。例えば De Jong のテスト関数⁴⁾のひとつとして知られる sphere function は、各遺伝子の自乗の総和を計算する適合度関数である。この場合、図 5 左のように記述すると関数 f に対する入力は過去と一致しないため再利用不可能であるが、図 5 右のように書きかえることで関数 f_2 について再利用可能となる。

ただし、サブ関数の入力遺伝子数をあまり多く仮定すると、入力が完全に一致する確率が低くなり、再利

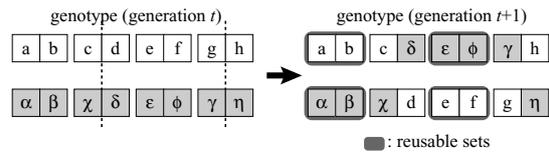


図 6 共通遺伝子の局所性と再利用可能性

用率も低く抑えられてしまう。また、入力遺伝子数を少なく仮定すると、サブ関数の処理量が少なくなってしまい、入力が一致した場合でも再利用の効果があまり上がらない可能性がある。このため、入力遺伝子数を適切に設定する必要がある。

4.3 前世代と共通する遺伝子の局所性

上述の関数分割を行う場合、再利用による効果は、サブ関数の入力とする遺伝子の選び方にも依存する。というのも、特に交叉方法が n 点交叉である場合、前世代の個体と共通している遺伝子集合には局所性があるためである。

入力となる遺伝子表現の全長を l 、構成する遺伝子を c_i ($1 \leq i \leq l$)、サブ関数が入力とする遺伝子数を m と仮定すると、遺伝子表現は s ($= l/m$) 個のサブセットに分割されて処理される。このとき連続した遺伝子が同じサブセットに含まれるよう、入力サブセット I_j ($0 \leq j \leq s-1$) を $I_j = \{c_{j \times m + 1}, \dots, c_{(j+1) \times m}\}$ と設定する必要がある。例えば 2 点交叉では、こうすることで s 個のサブセット中少なくとも $(s-2)$ 個は前世代で処理したものと全く同じとなり、高い再利用率が見込める。図 6 に示したのは $l=8$ 、 $m=2$ の単純な例であるが、この場合でも半数の処理が再利用可能となることが分かる。

これに対し、 $I_j = \{c_{j+1}, c_{j+s+1}, \dots, c_{j+(m-1)s+1}\}$ のように離れた遺伝子が同一入力セットとなるようなサブ関数を設定した場合、再利用率は大幅に低下してしまうため、注意が必要である。例えば図 6 と同じ例では、全ての入力サブセットが前世代と異なり、再利用不可能となってしまふ。

5. 評価

5.1 評価環境

評価には、再利用機構を実装した単命令発行の SPARC-V8 シミュレータを用いた。各パラメータを表 1 に示す。キャッシュ構成や命令レイテンシは、SPARC64-III⁵⁾ を参考にしている。再利用表については、まず IORB を一次キャッシュと同サイズの 32KB (32Byte 幅 \times 256 エントリ \times 4 セット) とし、レイテンシも一次キャッシュと同じと仮定した。L2RB については、2MB (256bit 幅 \times 64K エントリ) の CAM とし、レイテンシとしてレジスタとの比較に 32Byte/cycle、キャッシュとの比較に 32Byte/2cycle を仮定した。

またロードモジュールとしては、汎用 GA ソフ

表 1 シミュレーション時のパラメータ

D1 Cache 容量	32 KBytes
レイテンシ	2 cycles
ミス ペナルティ	10 cycles
D2 Cache 容量	2 MBytes
レイテンシ	10 cycles
ミス ペナルティ	100 cycles
ロードレイテンシ	2 cycles
整数乗算 "	8 cycles
整数除算 "	70 cycles
浮動小数点加減乗算 "	4 cycles
単精度浮動小数点除算 "	16 cycles
倍精度浮動小数点除算 "	19 cycles
IORB サイズ	32 KB
L2RB サイズ	2 MB
交叉率	60.0 %
突然変異率	0.1 %
個体数	50
適合度計算回数	1000 回

トウェアである GENE_sY_s⁶⁾ 1.0 を gcc 3.0.1 (-O2 -msupersparc) でコンパイルし、スタティックリンクにより生成したものをを用いた。古くから広く利用されている汎用 GA ソフトウェアに GENESIS⁷⁾ があるが、GENE_sY_s は GENESIS を機能拡張したプログラム集である。個体選択のスキームなどが拡張されており、適合度関数についても、ベンチマークテストや定量的評価によく用いられる De Jong のテスト関数をはじめとして、標準的な関数実装されている。

なお GENE_sY_s では、適合度計算のための遺伝子表現変換として、グレイコードの逆変換 (Degray) と、char 型配列から整数への変換 (Ctoi) を行っている。参考に、この部分のコードを図 7 に示す。

5.2 結果

評価モデルとしては、以下に示す 3 つを仮定し、比較を行った。各モデルが要した実行サイクル数を、GENE_sY_s が持つ 24 種の適合度関数 (f1 ~ f24) 全てについて測定した結果を図 8 に示す。なお、各関数の詳細については文献 6) を参照されたい。

(M) オリジナル (再利用なし)

(R) 再利用

(R') 4 章で示した改良を加えた上で再利用を適用

各適合度関数のグラフは、左から順に (M), (R), (R') が要した実行サイクル数を表しており、それぞれ (M) のサイクル数を 1 として正規化している。また凡例については、cross は交叉、mutate は突然変異、degray および ctoi は遺伝子表現変換、evaluate は適合度計算、select は個体選択の処理に要したサイクル数内訳をそれぞれ示している。なお misc は、初期化などのプロセスに要したサイクル数を示している。また GENE_sY_s は、各世代毎に個体の評価値平均などの計算を行っている。これは必ずしも GA に必要な処理ではないが、今回は GENE_sY_s における高速化を

```
void Degray(char* inbuf, char* outbuf,
            register int length) {
    register int i, last;
    for (last=0, i=0; i<length; i++) {
        if (inbuf[i]) outbuf[i] = !last;
        else outbuf[i] = last;
        last = outbuf[i];
    }
}

int Ctoi(register char* buf, register int length) {
    register int i, answer;
    for (answer=0, i=0; i<length; i++) {
        answer <<= 1;
        answer += *buf++;
    }
    return(answer);
}
```

図 7 GENE_sY_s の遺伝子表現変換関数

評価するという意味で、この処理に要する時間もそのまま残した。これをグラフ中では measure としている。全ての凡例は、入力一致比較などの再利用に要したオーバーヘッドを含んだサイクル数を表している。

グラフから、適合度関数の計算部分である deggray, ctoi, evaluate が再利用の適用により大幅に削減されていることが分かる。はからずも GENE_sY_s のソースコードが、4 章で述べたような手法にある程度沿っていたものに関しては、(R) でも高い効果が得られている。また、(R) では再利用の効果が低いものに関しても、(R') によって効果が向上している。結果、(R) でも最大 83%、平均 28% と高いサイクル数削減率が得られたが、(R') により更にこれを最大 87%、平均 39% まで引きあげることができた。f12 や f14 など、再利用による効果がほとんど得られなかったものに関しても、再利用オーバーヘッドによる性能低下は発生しなかった。

また、f5, f11, f13, f16, f17 など、(M) において evaluate の割合が高い適合度関数、すなわち処理全体に要する時間が他よりも大きい関数に対し、より効果的に再利用が働く傾向があることにも注目されたい。なお、この 5 関数における (R') の平均サイクル数削減率を算出したところ、57% という値になった。

5.3 考察

ここで、再利用による高速化に対し、GA のパラメータが与える影響について考える。

GENE_sY_s では、前世代から変化のなかった遺伝子表現、すなわち交叉および突然変異の対象とならなかつた遺伝子表現に関しては、適合度計算を省略する。よって仮に交叉率を変更した場合でも、4.3 節で示した再利用率に対する影響はなく、今回の結果とほぼ同じ結果が得られると考えられる。これに対し、突然変異率が大きくなった場合は再利用率が低下すると考えられるが、一般に GA では突然変異率をあまり大

なお、GENE_sY_s の f11, f12, f14 においては、Degray/Ctoi による遺伝子表現変換の処理は行われない。

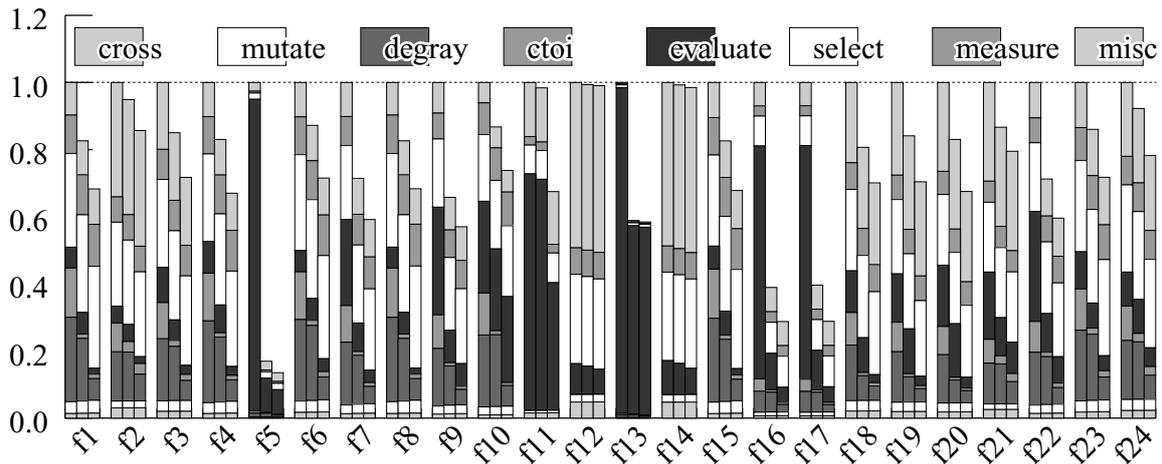


図 8 実行サイクル数の比較 (GENEsYs)

きな値に設定することはないため、プログラム全体の速度に対する影響は軽微であろう。また適合度の評価回数は、増大するに従い全体における misc の占める割合が低下し、再利用対象となる適合度計算の割合が増加するため、再利用の効果も大きくなる。

個体数に関しては、大きくなるに従って遺伝子表現パターンも多種となり、再利用率を維持するためには適合度関数の入力パターンをより多く保存できる必要がある。このため、L2RB を構成する CAM 容量が許す範囲で再利用率は維持されるが、L2RB 溢れが発生するとある程度の再利用率の低下が予想される。交叉方法に関しては、 n 点交叉の場合、あまり n が大きくなると再利用率はある程度低下すると考えられるが、遺伝子表現長が十分長ければその影響は少ない。

6. おわりに

本稿では GA に対し、並列 GA とは異なる高速化のアプローチとして、再利用を適用することによる手法を提案した。GA の処理プロセスについて概説し、一般に最も時間を要する処理である適合度の算出に対して、再利用が効果的であることを述べた。再利用の際、入力となる遺伝子表現のアドレスの違いを吸収するためにバッファリングが必要であることを述べた。また n 点交叉では、遺伝子表現が前世代の遺伝子表現と多くの共通部分を持つことから、適合度計算の関数を分割することで、再利用率を向上させることができると示した。

汎用 GA ソフトウェアである GENEsYs を用いて再利用の効果の評価した結果、24 種全ての適合度関数において平均 28%、最大 83% のサイクル数を削減することができることを示した。更に、適合度関数の分割を行うことにより、これを平均 39%、最大 87% まで引きあげることができた。また、全体の処理時間

に占める適合度関数の割合が高い、本来最も高速化が望まれる関数において、再利用の効果が高いたことを示し、そのような 5 関数で平均 57% のサイクル数を削減できることを示した。

なお、本稿で提案した高速化手法は、従来 GA の高速化手法として多く用いられてきた並列 GA と競合するものではない。よって、双方を組み合わせることによって、更なる高速化が可能となると考えられる。

参考文献

- 1) Cantú-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*, Genetic Algorithms and Evolutionary Computation, Vol. 1, Kluwer Academic Publishers (2000).
- 2) Lobo, F. G., Lima, C. F. and Mártires, H.: An Architecture for Massive Parallelization of the Compact Genetic Algorithm, *Proc. of the Genetic and Evolutionary Computation Conference* (2004).
- 3) 津邑公暁, 中島康彦, 五島正裕, 森眞一郎, 富田眞治: 並列事前実行機構における主記憶値テストの高速化, 情報処理学会論文誌: コンピューティングシステム, Vol. 45, No. SIG 1(ACS 4), pp. 31-42 (2004).
- 4) Jong, K.D.: *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, PhD Thesis, University of Michigan (1975).
- 5) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).
- 6) Bäck, T.: GENEsYs 1.0. Software distribution and installation notes (1992).
- 7) Grefenstette, J.J.: GENESIS: A System for Using Genetic Search Procedures, *Proc. of the 1984 Conference on Intelligent Systems and Machines*, pp. 161-165 (1984).