社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－MPS－46 (17)
2003／9／19

自己適応島GAの非同期型並列実装

高島栄一[†] 村田佳洋[†]
柴田直樹[†] 伊藤実[†]

筆者らが以前提案した自己適応島GAは，各島のパラメータをやりとりする際に全ての島が同期して動作しなければならず，島を担当する計算機の能力に違いがある場合に待ち時間が発生していた．本手法では，アルゴリズムに改良を加え，同期を取る必要をなくすることにより性能を向上させた．また，比較実験を通して，計算機の能力に違いがある場合に，従来アルゴリズムよりも時間あたりの探索性能が改善されていることを確認した．また，一定評価回数での探索性能を比較し，解の探索能力が若干改善されていることを確認した．

Asynchronous Implemention of Self Adaptive Island Genetic Algorithm

Eiichi Takashima,[†] Yoshihiro Murata,[†] Naoki Shibata[†] and Minoru Ito[†]

We have previously proposed SAIGA(self adaptive island GA), but it requires all islands to be synchronized when exchanging parameters between islands. In this paper, we propose a technique to avoid this synchronization. We also confirmed that our new algorithm largely outperforms our previous algorithm if there are large differences between processing power of each island. Through experiments, we confirmed that there is slight improvement of search performance from our previous algorithm if both of algorithm uses same number of evaluations.

1. はじめに

遺伝アルゴリズム (GA) は自然界の進化を模倣した組み合わせ問題に対する最適化手法である．GAを使って問題の近似解を求めるためには，交叉率，突然変異率等のパラメータ値の設定を行う必要があり，解の探索効率はこれらのパラメータ値に強く依存する．ところが，人手によってパラメータ値の調整を行うことは，非常に手間のかかる作業である．この理由として，最適なパラメータ値が，取り扱う最適化問題，交叉手法，突然変異手法等に依存すること，また，最適なパラメータ値がパラメータ同士で互いに独立でないことが挙げられる．そこで，自動的にパラメータ値を適応させる手法として，自己適応GA(Self Adaptive GA)[1] が研究されている．我々の研究グループでは，メタGA[2] と環境分散型並列GA[3] の長所を取り入れた，自己適応島GA (Self Adatptive Island GA, 以下SAIGA)[4] を提案している．

SAIGAでは，各島のパラメータのやりとりの際に，全ての計算機の間で同期をとる必要があった．このため，各計算機の計算能力に差がある場合，並列化による効果を十分生かすことができなかった．

本稿では，SAIGAに対して，パラメータの扱いを工夫することにより，非同期でも動作できるように改良を加えたA-SAIGA (Asynchronous-SAIGA) を提案する．

以下，この論文では，2章で従来手法，3章で提案手法について述べる．また4章で実験について述べ，5章で実験結果ついて考察し，最後に，結論と今後の課題について述べる．

2. 従来手法

従来手法であるSAIGAは島GAの一種であり，メタGAの仕組みを用いてパラメータ適応を行う．ここで，それぞれの島で扱うGAをlow level GAと呼ぶ．これらの島で用いられるパラメータ値を，high level GAと呼ばれる別のGAの個体として扱う．SAIGA

† 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

の島はそれぞれ，独自の個体群とパラメータ値をもっている．SAIGAはメタGAと同様に，low level GAで解探索を行い，high level GAでパラメータ値の適応を行う．

SAIGAを用いた場合，low level GAのパラメータ値の調整をする必要がなくなる一方で，high level GAのパラメータ値の調整が必要になる．しかし，low level GAの扱う問題が変わっても，low level GAのパラメータの組合せ空間が変わらなければ，パラメータ値を探すという，high level GAにとっての問題は基本的に不変である．したがって，適切なhigh level GAのパラメータ値がわかれば，low level GAが解く問題に依存せず，この設定を使うことができる．

SAIGAの各島での探索は，ある一定の評価回数分の探索（時代と呼ぶ）を単位として行われる．一時代あたりの評価回数は，全ての島の探索において一定であり，あらかじめ与えられる．それぞれの島が，1時代分の評価回数をいかに利用するかは，各島の持っているパラメータ値によって変わる．例えば1時代の評価回数が100である場合，1世代あたりの評価回数が20であれば，20個体を5世代分進化させる．

high level GAの個体（各島の持つパラメータ）に対して，1時代ごとに，評価値が与えられる．評価値は，時代の最後のエリート個体の評価値から時代の最初のエリート個体の評価値を引いたもの（但し移民によるエリート個体の評価値の変化は除外）である．この評価値に基づいて，high level GAにおける個体群に対して遺伝演算子が適用され，パラメータ値が進化する．high level GAの新たな世代におけるパラメータ値を各島に割り振り，次の時代のlow level GAの探索を行う．この過程を繰返すことにより，SAIGAでは解の探索とパラメータ値の適応を同時に行う．

SAIGAは構造上，島に対して与えられるすべてのパラメータ値を適応させることができるが，本稿では，1世代あたりの評価回数，淘汰圧，交叉率，突然変異率の4つのパラメータ値を適応させている．最初，これらのパラメータ値はランダムに与えられる．

3. 提案手法

3.1 概　　要

本提案手法では，high level GAの個体群を時代毎に全て更新するのではなく，low level GAの一つの島の時代が終わる毎に一個体ずつ更新する．low level GAの時代が終わると，そのパラメータを新しい個体として追加し，high level GAの個体群から，最も古いものを一つ削除する．次に，high levelの個体群から，評価値をもとに2つの個体を選択し，これらの間で交差と突然変異を行い，新しい個体を得て，このパラメータを使用してlow level GAの時代を開始する．

3.2 詳　　細

3.2.1 記号の定義

提案手法のアルゴリズムの説明をする前に，定義を行う．

組み合わせ最適化問題の探索空間の中のある一つの点の座標ベクトルと実数の評価値の二つ組みを個体とする．

vは，GAで用いるパラメータベクトルである．また，high level GAで用いるパラメータベクトルを特にv_hと表記する．

SAIGAで用いる個体をpとする．pは，染色体$p.x$，評価値$p.fit$により構成される．low level GAの個体をおよび個体群を$p_l \in P_i$，high level GAの個体および個体群を$p_h \in P_h$とする．ここで，iは島に割り振られた番号である．

$p_h.x \in X_h$から，パラメータベクトル$v \in V$を与える関数を$f : X_h \to V$を定義する．また，その逆関数である$f^{-1} : V \to X_h$を定義する．

A-SAIGAには，以下のパラメータが与えられる．

- **evaluation_counter_per_era** :
  時代毎に与えられる評価回数
- **island_max** : 島の数
- v_h : high level GAのパラメータベクトル

3.2.2 アルゴリズム

各島で用いられるlow level GA関数であるlowlevelGA$(v, P_i, p_l^{\mathbf{mig}})$は，**evaluation_counter_per_era**の評価回数を用いて個体群P_iを進化させる関数である．入力はパラメータベクトルv，進化前の個体群P_i，他の島から送られた移民個体$p_l^{\mathbf{mig}}$であり，出力は進化後の個体群P_i，パラメータベクトルvに対する適応度fit，他の島に送る移民個体$p_l^{\mathbf{mig}}$である．

Algorithm all_start()
1 parameter_create() プロセスの起動;
2 parameter_receive() プロセスの起動;
3 **for** $i := 1$ to **island_max do**
4 　island(i) プロセスの起動;
5 **next**

Algorithm island(i)
1 島iの個体群P_i内の個体$p_l \in P_i$をすべて初期化;
2 $p_l^{\mathbf{mig}}$を初期化; // 移民のための個体
3 パラメータベクトルvをランダムに生成;
4 $(P_i, fit, p_l^{\mathbf{mig}}) :=$ lowlevelGA$(v, P_i, p_l^{\mathbf{mig}})$; // ここで$fit$はパラメータベクトル$v$に対して返された評価値.
5 parameter_receive() プロセスに対して，$(v, fit, p_l^{\mathbf{mig}}, i)$を送る;
6 **while** true **do**

```
7       repeat
            parameter_create に対して，島番号 i を送る;
8       until parameter_create() プロセスから (v, p_l^mig)
        を受け取った
9       (P_i, fit, p_l^mig) := lowlevelGA(v, P_i, p_l^mig);
10      parameter_receive() プロセスに対して，(v, fit, p_l^mig, i)
        を送る;
11  endwhile
```

Algorithm parameter_create()

```
1   while true do
2       j=0;
3       while true do
4           if island(i) プロセスから島番号 i を受け取った
            then
5               break ;
6           endif
7       endwhile
8       while P_h = ∅ do
9           delay(); // P_h に要素が追加されるまで待機
10      endwhile
11      p_h^parent1 := selection(P_h, v_h); //選択により親個
        体を得る
12      p_h^parent2 := selection(P_h, v_h);
13      p_h^offspring := crossover(p_h^parent1, p_h^parent2, v_h);
        //交叉により子個体を得る
14      p_h^offspring := mutation(p_h^offspring, v_h);
15      v := f(p_h^offspring.x); //パラメータベクトルに変換
16      If |P_i^mig| ≥ 1 then
17          p_l^mig := best_of(P_i^mig); //島 i に送られた最
            も評価値の高い個体を抽出する
18          P_i^mig := ∅;
19      else
20          p_l^mig にダミーの値を代入する;
21      endif
22      island(i) プロセスに (v, p_l^mig) を送る.
23  endwhile
```

Algorithm parameter_receive()

```
1   P_h := ∅;
2   P_i^mig := ∅;
3   while true do
4       if island(i) プロセスから (v,fit,p_l^mig,i) を受け取っ
        た then
5           if |P_h| < p_h^max then
6               P_h := P_h − {oldest_of(P_h)};//最も古く
                からある要素を削除
7           endif
8           p_h.x := f^{-1}(v);// パラメータベクトルを染色
            体に逆変換
9           p_h.fit := fit;
10          P_h := P_h ∪ {p_h};
11          if i = island_max then
12              P_0^buf := P_0^buf ∪ {p_l^mig};
13          else
14              P_{i+1}^buf := P_{i+1}^buf ∪ {p_l^mig};
15          endif
16      endif
17  endwhile
```

4. 実験と考察

4.1 実　　験

A-SAIGA の探索効率を調べるために，以下の条件で実験を行った．**island_number** を 10 とし，10 のプロセッサを用いた (**island_number**)．実験に用いたプロセッサの構成は，2.4GHz × 3 台，2.0GHz × 1 台，1.5GHz × 2 台，800MHz × 3 台，200MHz × 1 台である．島毎にプロセッサを割り当て，その島のプロセッサのうち，1 つのプロセッサで all_start() プロセス，parameter_create() プロセス，parameter_receive プロセス () を動作させた．また，比較対象として，従来手法である SAIGA を用いた．

High level GA の遺伝演算子として，二点交叉，ルーレット選択を用いた．また，交差率は 0.8，突然変異率は 0.6，スケーリング率を 1 対 5 とした．

評価実験の評価関数として，巡回セールスマン問題である 105 都市問題 lin 105[5] を用いた．最適値は 14379 である．突然変異として 2opt を使用し，交叉方法として，EXX[6] を使用した．突然変異率は，都市ごとに突然変異を行う確率とした．

4.2 結　　果

評価回数単位の評価値の変化を，評価回数単位の変化と呼ぶ．図 1 は，評価回数の増加に伴う評価値の推移を示したグラフである．このグラフから，評価回数単位の変化について提案手法は，従来手法に比べて若干改善されていることが読み取れる．

この原因について，以下のような考察を行った．SAIGA では，島間の同期がとられているためにパラメータベクトルが 10 個評価される間に，すべての島において，ちょうど 1 時代分の探索が平等に行われる．しかし，A-SAIGA では，パラメータベクトルが 10 個評価される間に，速いプロセッサが割り当てられた島において，複数時代分の探索が行われることがある（逆に，島によっては，1 時代分の探索も行われないことがある）．この局所的な探索により．改善がなされたと考えられる．しかし，並列に用いるプロセッサの計算能力の組合わせ，また，取り扱う問題によって，必ずしも，評価回数単位の変化について，改善されるとは限らない．このことについて，今後，さらに実験を重ねていく必要がある．

経過時間単位の評価値の変化を，時間単位の変化と呼ぶ．図 2 は，実行時間の増加に伴う評価値の推移を示したグラフである．この時間は実測値である．このグラフから，時間単位の変化について提案手法は，従来手法に比べて大きく改善されていることが読み取れ

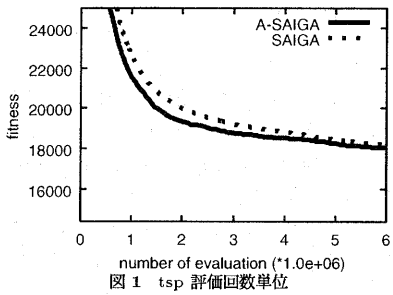

図 1　tsp 評価回数単位

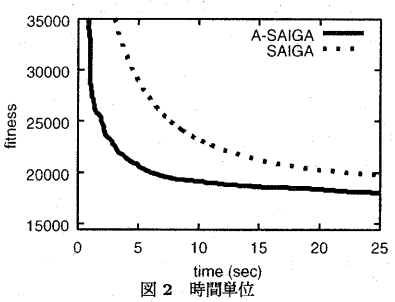

図 2　時間単位

る．これは，提案手法で目指したとおり，同期のために浪費されていた計算機の空き時間を有効活用することが行われたためであると考えられる．

また，従来手法と提案手法のそれぞれに対し，$6.4 * 10^6$ の評価回数の探索を行うために，かかった時間の実測を行った．その結果，従来手法では，70.88 秒，提案手法では，27.26 秒かかった．その比率は 0.38 倍である．この結果から，時間の有効活用することが行われていることがわかる．一定の時間において，提案手法は，従来手法より，より多くの評価回数を用いた探索を行うことを示している．

5. おわりに

本稿では，並列動作する際に同期する必要があった従来手法に対し，high level GA での個体群の扱い方を工夫することにより同期する必要をなくした A-SAIGA を提案した．

従来手法は同期のために計算時間を浪費する特性を持っていたが，提案手法はこのような浪費をしないために，同じ計算時間でより多く計算することができる．

また，比較実験により，計算時間の浪費の除去による探索効率の向上を確かめた．

今後の課題として，TSP 以外の問題に対して実験を行うこと，島に与える評価回数と島数の関係について調査すること，計算機間の計算能力の差が結果にどのような結果を与えるかを調べることが挙げられる．また，現在は high level GA の構造を集中管理方式にて実現しているので，分散管理の手法を確立することによりスケーラビリティを考慮することが挙げられる．

参考文献

1) Bäck, T. Self-adaptation in genetic algorithms. In F.J.Varela, P. B., editor, Proceedings of 1st European Conference on Artificial Life, pp. 263–271, (1992).
2) Weinberg, R. Computer simulation of a living cell. Dissertations Abstracts International, 31(9), 5312B, (1970).
3) Miki, M., Hiroyasu, K., Kaneko, M and Hatanaka, I. A Parallel Genetic Algorithm with Distributed Environment Scheme. IEEE Proceedings of Systems, Man and Cybernetics Conference SMC'99, pp. 695–700, (1999).
4) Takashima, E, Murata, Y. Shibata, N. and Ito, M. Self Adaptive Island GA. 2003 IEEE Congress on Evolutionary Computation に投稿, 採録決定.
5) http://www.iwr.uni-heidelberg.de/groups/ comopt/software/TSPLIB95/
6) Maekawa, K. Mori, N. Tamaki, H. Kita H. and Nishikawa, H. Genetic Solution for the Traveling Salesman Problem by Means of a Thermodynamical Selection Rule. Proceedings 1996 IEEE International Conference on Evolutionary Computation, pp.529–534, (1996).