ハードウェア化のための多目的GAアーキテクチャの提案

橘 達弘[†1] 村田佳洋[†1] 柴田直樹[†2]
安本慶一[†1] 伊藤実[†1]

多目的遺伝的アルゴリズム（Multi-Objective Genetic Algorithms，MOGA）は，多目的最適化問題を解くために単一目的遺伝的アルゴリズムを拡張した最適化手法である．MOGAでは複数の個体群の多様性を維持するための手法であるニッチ法やランク戦略がよく用いられるため，単一目的GAよりさらに計算量が大きくなる傾向がある．本論文では，多目的最適化問題を高速に解くことを目的とし，ハードウェア化のためのMOGAのアーキテクチャを提案する．提案方式では，世代交代モデルとしてハードウェア化に適したMinimal Generation Gapモデルを採用する．既存のニッチ法やランク戦略をパイプライン処理で実装することは困難なため，パイプライン処理に適した多様性を維持する手法を設計，採用した．また，解探索能力の向上のために，島モデル型GAの各島の目的関数を改変した並列GAモデルに即した並列実行方式を設計し，提案アーキテクチャに採用した．実験の結果，提案アーキテクチャによるMOGA回路はNSGA-IIより優れた探索能力を持つことを確認した．

Proposal of a MOGA for Hardware Implementation

Tatsuhiro Tachibana,[†1] Yoshihiro Murata,[†1] Naoki Shibata,[†2]
Keiichi Yasumoto[†1] and Minoru Ito[†1]

Multi-Objective Genetic Algorithms (MOGAs) are enhancement of Single-Objective Genetic Algorithms (SOGAs) to solve multi-objective optimization problems. Since MOGAs require a special selection mechanism such as ranking strategy and niching method to preserve diversity of individuals, MOGAs require larger computation power than SOGAs. In order to improve calculation speed of MOGAs, we propose a new method to easily implement MOGAs as high performance hardware circuits. In the proposed method, we adopt a simple minimal generation gap model as the generation model, which is easy to be pipelined. Since it is difficult to implement niching method and ranking strategy as pipelined circuits, we developed a new selection mechanism which is suitable for hardware implementation. In order to improve search efficiency, our method also includes a parallel execution architecture based on island GA. In this architecture, we use different objective function for each island. Through experiments, we confirmed that our method has higher search efficiency than NSGA-II.

1. はじめに

多目的最適化問題（Multi-Objective Optimization Problem）とは，複数の異なる目的に対し，パレート最適解と呼ばれる複数の最適解の集合を探索する問題である．多目的最適化問題は，飛行機，エンジンなどの設計の際に，相反する要求を満たすパラメータを求める問題として用いられる[1)–3)]．多目的最適化問題を解く手法の1つに多目的遺伝的アルゴリズム（Multi-Objective Genetic Algorithm，MOGA）がある．MOGAは，組合せ最適化手法の1つである単一目的遺伝的アルゴリズム（Genetic Algorithm，GA）を多目的最適化問題に適用できるように拡張したものである．GAは，生物の進化を模倣した最適化手法で，解候補を個体上の染色体に持たせ，複数の個体を用いた多点探索を行う．GAは新たな問題に適用するのが容易である一方，専用のアルゴリズムと比べて計算量が大きいことが知られている．GAを高速に実行する手法としてハードウェアGAが研究されており，単一目的GAのハードウェア化による高速化は複数報告されている[4)–9)]．

本論文では，多目的最適化問題を高速に解くために，ハードウェア化に適したMOGA回路を実装するため

†1 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
†2 滋賀大学経済学部
Faculty of Economics, Shiga University

のアーキテクチャを提案する．このアーキテクチャは，ハードウェア化に適した MOGA と，そのモジュール単位での実装手法を含む．これらの実現には以下の 2 つの問題点がある．

（1）MOGA は，複数の解候補となる個体を保持し，それらに遺伝的操作を適用し最適解を探索する．MOGA で一般に用いられている世代交代モデルでは，MOGA が保持している個体群を基に新世代の個体群をすべて生成し終えるまで，GA の操作の 1 つである選択操作を行えない．そのため，パイプライン処理での実装が困難である．

（2）MOGA では複数のパレート最適解を効率良く探索するために，解の多様性を維持するニッチ法[10]やランク戦略[11),12)]などが用いられる．しかし，ニッチ法やランク戦略はすべての解候補（個体）間で比較を行うためにソートなどの繰返しをともなう操作を必要とし，ハードウェア化に不向きである．

提案手法では，問題点（1）を解決するため，簡易化した Minimal Generation Gap（MGG）[16] モデルを，問題点（2）のためにパイプライン処理に適した多様性維持機構を新たに設計し採用する．さらに，解探索能力の向上と回路の並列拡張性を確保するために島モデル型 GA を拡張し，島ごとに異なる部分のパレート最適解を探索させる MOGA に適した並列実行または並列化手法を採用する．

提案するアーキテクチャは様々な多目的最適化問題に対して適用することが可能である．またモジュール化されているため，交叉や突然変異などの遺伝演算子をモジュール単位で交換し，効率の良いものを用いることができる．

提案アーキテクチャは，MGG モデルを利用した世代交代モデルと独自の重複排除機構を持つため，一般的な MOGA とは特性が異なる．提案アーキテクチャによる MOGA 回路の有効性を評価するために，代表的な MOGA の 1 つである NSGA-II [11] との性能比較を行った．対象問題には多目的ナップサック問題を適用した．実験の結果より，提案手法が NSGA-II より優れた解探索能力を持つことを確認した．

2. 関 連 研 究

GA のハードウェア化による高速化手法として以下のものが提案されている．文献 4) では，若林らが交叉法に適応選択を組み込んだ単一目的 GA をハードウェア化している．文献 5) では，小林らが不連続閉曲線抽出手法を対象としたハードウェア GA を実装している．これらの研究では，ハードウェアで実装した単一目的 GA の計算速度を向上させるために，どのようにストールの発生を防ぐかに重点がおかれている．文献 6) では H^3 エンジンと呼ばれる単一目的 GA 専用ハードウェアを比較的ストールの発生しにくい steady-state GA で実装している．Shackleford らは文献 7) で，survival based steady-state GA を比較的容易な問題である集合被覆問題に適用している．Aporntewan らは文献 8) で，steady-state 型の単一目的 GA である Compact Genetic Algorithm（CGA）を one-max 問題に適用している．橘らは文献 9) で，パイプライン化と並列化が容易である GA 回路のためのアーキテクチャを提案している．これらの既存の研究は単一目的の GA をターゲットとしており，著者らの知るかぎりにおいて MOGA をハードウェア化した研究は見当らない．

また，ソフトウェアで実装した MOGA の研究には以下の例がある．Deb らは文献 11) で，Non-Dominated Sorting Genetic Algorithm-II（NSGA-II）を，Zitzler らは文献 12) で，Strength Pareto Evolutionary Algorithm-II（SPEA-II）を提案している．これらの手法は，それぞれ独自のランク戦略を用い，優れた性能を示している．しかし，これらは機構が複雑であるためハードウェア化には適さない．

3. 提 案 手 法

MOGA では多様なパレート最適解の探索が必要なため，単一目的 GA 向けの高速実行アーキテクチャ[9] をそのまま利用して多目的最適化問題を解くことは困難である．本章では，多目的最適化問題の定義，MOGA のハードウェア化の問題点と解決法を述べた後，MOGA に適したモジュール単位での実装手法，およびその並列化手法を提案する．

3.1 多目的遺伝的アルゴリズム

多目的最適化問題では，同時に複数の目的を最適化する必要があるため，1 つの目的に対する適応度の比較では解の優劣を決められない．解候補である個体の優劣は以下のように定義される．

個体 x，y を解候補とする．目的の集合 P の要素 p に対する目的関数値が，それぞれ $f_p(x)$，$f_p(y)$ で表されるとする．目的関数値は，その個体がその目的においてどれだけ良いかを表す値である．ここでは，目的関数値が大きいほど良いとする．以下の式が成立するとき，個体 x は個体 y を**支配する**といい，個体 x は個体 y より優れているとする．

$$\exists_{p\in P}(f_p(x) > f_p(y)) \wedge \forall_{p\in P}(f_p(x) \geq f_p(y))$$

パレート最適解は解空間において他のどの解にも支配されない解を指す．一般にパレート最適解は解空間

中の点の集合として表現される．多目的最適化問題はすべてのパレート最適解の集合（パレートフロント）を見つける問題である．

MOGA は，多目的最適化問題を解くために，解候補である**個体**を複数用いて近似解を求める．個体は，**染色体**（chromosome）と各目的の目的関数値（適応度）からなる．染色体は，与えられた問題の探索空間内の 1 点（解候補）を表す．代表的な世代交代モデルを用いた MOGA のアルゴリズムを以下に示す．

（i）染色体を乱数で生成した個体を，多数作成する．（ii）各個体の適応度を計算する（**評価操作**）．（iii）各個体の支配・被支配関係を調べ，支配されていない個体を優先的に残し，残りの個体を破棄する（**選択操作**）．（iv）個体群から 2 つの個体を取り出しペアを作成する．これらを両親（parent）とし，両親の性質を混ぜ合わせて子個体を作成する（**交叉操作**）．（v）新しい個体の染色体に対しある確率で突然変異を起こす（**突然変異操作**）．（vi）ステップ（ii）に戻る．

MOGA はこれらの操作を指定した時間（または回数）繰り返し適用することでパレート最適解を探索する．MOGA のハードウェア化のためには，以下の 2 つの問題点がある．（1）上記の世代交代モデルは，選択操作の際に親の個体群とそれと同数の子の個体群をすべて保持する必要があるため，ハードウェア化の際に回路規模が増大する．（2）MOGA は単一目的 GA と異なり，選択操作の際に多様性を維持することが重要である．NSGA-II [11] では選択操作の際に，多様性維持機構としてランク戦略と混雑度（crowding distance）を使用する．ランク戦略は，個体にランクをつけることで個体間の優劣を決定する．混雑度は，同ランクに属する個体間の優劣を決定する．これらの手法を実行するためには，親個体群と子個体群の全体の中から個体間の優劣を決定するために，この全個体群内の個体間で繰り返し比較を行う必要がある．そのため，他の遺伝的操作を同時に行うことができず，パイプライン処理によるハードウェア化は困難である．

3.2 提案アーキテクチャ

問題点（1）は MOGA 特有の問題ではなく，単一目的 GA でも同様である．そこで，文献 9) で提案した，Minimal Generation Gap（MGG）モデル[16] を利用した世代交代モデルを本方式にも採用することで，パイプライン処理に適したハードウェア化を実現する．また，問題点（2）を解決するために，パイプライン処理に適した多様性維持機構である重複個体排除機構を提案する．さらに，解探索の性能を向上させるため，MOGA に適した並列化手法も提案する．

3.2.1 提案手法における世代交代モデル

MGG モデル[16] では，交叉・突然変異操作の際，個体群中より選択された 2 つの親個体より 2 つの子個体のみ作成する．そして子個体がより優れていれば親個体と置き換えられる．提案アーキテクチャでは，2 つの親個体から 1 つの子個体のみを作成する．そのため，現個体群のほかには，これら 3 つの個体のみを保持すればよい．よって，回路規模を抑えることができ，パイプライン処理による実装が容易である．

3.2.2 重複個体排除機構

提案アーキテクチャでは個体群の多様性を維持するため，重複した個体の排除を基本とした手法を採用する．

本項で提案する重複固体排除機構は，重複しているあるいは支配されている個体を発見して，それを重複しておらずかつ支配されていない個体に置き換える．またこの置き換えが発生しなかったときには，選択機構における結果を反映して親個体を子個体に入れ替える．ここで，実数値を取り扱う問題などでは，細部が異なるため完全には重複しないが，類似した個体が多数発生すると考えられる．このような問題においては，ニッチ法[14] の考えを応用する．ここでは解空間における適応度間のマンハッタン距離を計算し，それが与えられた定数よりも小さい 2 個体は重複していると見なす．

提案機構は，パイプライン処理で実装するため，子個体 c とすべての現個体群 I に対して 1 つずつ順番に比較を行う．比較の結果，子個体とある現個体の適応度が同一のとき，それらは重複していると見なす．このとき，「どちらか（あるいは両方）の個体を排除するべき」と見なして個体 c および比較された個体に排除のためのフラグを立てる．最終的に重複個体中から 1 つだけを残すように，パイプライン中で重複個体を発見するたびにフラグを立てていく（詳細は後述のアルゴリズムを参照）．同様に，パイプライン中で支配関係が発見されたときには，c に支配された個体にフラグを立てる．フラグの立てられた子個体は，現個体群を置き換えることなくパイプラインを流れてゆき（重複個体ががあればフラグを立てる操作は行う），フラグの立てられた親個体は，フラグの立てられていない子個体によって置き換えられて消滅する．これらのルールに従ってパイプライン中に子個体を流し続けることによって，現個体群から重複個体を排除する．ただし，重複個体排除機構を以下のように実装する（**図 1** 参照）．k 個のサブモジュールを用意する．ここで k は MOGA で保持する個体数であり，MOGA は

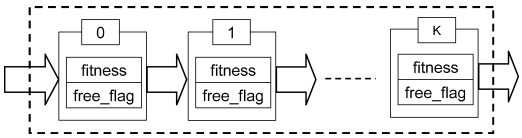

図 1 重複個体排除機構
Fig. 1 Overlap rejection mechanism.

現世代の個体群（現個体群）$Y = \{y_1, ..., y_k\}$ を保持している．このため，提案手法はサブモジュールよりも多くの個体数を取り扱うことはできない．各サブモジュールは 3 つのパラメータを持っている．それぞれ，サブモジュールの識別番号 i，個体群中の個体 y_i の適応度 $fitness_i^y$，個体 y_i が個体群中で重複するかどうかを示すフラグ $free\_flag_i^y$ である．各サブモジュールはパイプライン処理される．i 番目のサブモジュール S_i は，$i-1$ 番目のサブモジュール S_{i-1} から情報を受信する．ここでは，各サブモジュールが逐次受信する新しい子個体の集合を $X = \{x_1, ..., x_k\}$ とする．サブモジュール S_i は以下の情報を受信する．子個体 x_i の適応度（$fitness_i^x$）と染色体（$chromosomes_i^x$），アドレス（$overwrite\_address_i^x$），選択操作による子個体と親個体を入れ替えるかどうかを表すフラグ（$selected\_flag_i^x$），個体群中に重複個体を発見したかを示すフラグ（$found\_flag_i^x$）である．ここではアドレス（$overwrite\_address_i^x$）の初期値は親個体のアドレスであり，各フラグの初期値は false である．

サブモジュール S_i のアルゴリズムを**図 2** に示す．

図 2 の (2) から (4) は重複個体または親個体が支配されていたことによる上書きのためのフラグ立てを行う．(5) から (6) は，MGG による個体の置き換えを行う．(7) は親個体と子個体が重複個体であるときの操作を行う．(8) から (10) は親個体にフラグが立っているときの子個体の上書き操作である．

3.2.3 MOGA に適した並列化手法

文献 9) では，回路の並列化による最大動作周波数の減少を防ぐために島モデル型 GA（Island GA，IGA）を基にした並列化手法を採用している．この手法では隣り合う 2 つの島間のみで，個体のやりとりを行うことで，機構を簡易化し，クリティカルパスの長さを抑え，最大動作周波数の減少を抑えている．しかし，この手法をそのまま MOGA に適用した場合，すべての島が解空間中に存在するパレート最適解の同一部分のみ探索する恐れがある．また，多目的最適化問題を効率良く解くための手法として島ごとに異なる目的を優先して探索する手法も提案されている[17]．

提案アーキテクチャでは，文献 9) の並列化手法に対し，島ごとに解空間中の異なる部分のパレート最適解を探索するよう改変を加えることで効率良く解探索を行う手法を採用する．

(1) 入力として，サブモジュールは，サブモジュール S_{i-1} から以下のパラメータセットを受け取る．
$fitness_i^x$, $chromosome_i^x$, $selected\_flag_i^x$, $overwrite\_address_i^x$, $found\_flag_i^x$.
(2) If $found\_flag_i^x$ == false then goto 5.
(3) If (個体 x_i が個体 y_i を支配，または 個体 x_i と個体 y_i の適応度が同一) then $free\_flag_i^y \leftarrow$ true.
(4) Goto 11.
(5) If not ($selected\_flag_i^x == true$ and $(i == overwrite\_address_i^x)$ then goto 7.
(6) $fitness_i^y \leftarrow fitness_i^x$,
$free\_flag_i^y \leftarrow$ false,
$found\_flag_i^x \leftarrow$ true,
goto 11.
(7) If (個体 y_i と個体 x_i がの適応度が同じ) then
  If $found\_flag_i^x$ == false then
    $found\_flag_i^x \leftarrow$ true,
    $selected\_flag_i^x \leftarrow$ false,
    goto 11.
  else
    $removed\_flag_i^y \leftarrow$ true,
    goto 11.
  else goto 8.
(8) If (個体 x_i が個体 y_i を支配) then
  $free\_flag_i^y \leftarrow$ true.
(9) If ($free\_flag_i^y$ == false or $selected\_flag_i^x$ == true) then goto 11.
(10) $fitness_i^y \leftarrow fitness_i^x$,
$free\_flag_i^y \leftarrow$ false,
$found\_flag_i^x \leftarrow$ true,
$selected\_flag_i^x \leftarrow$ true,
$overwrite\_address_i^x \leftarrow i$.
(11) 出力として，次のサブモジュール S_{i+1} に子個体 x_i のパラメータセットを渡す．

図 2 重複個体排除アルゴリズム
Fig. 2 Overlap rejection algorithm.

提案手法による並列 MOGA 回路は l 個の特定の目的を優先して探索する**偏向 MOGA** と 1 個以上の一般的な MOGA（**通常 MOGA**）からなる．l は対象となる問題の目的の数である．通常 MOGA と偏向 MOGA はそれぞれ**通常選択**と**偏向選択**を用いる．通

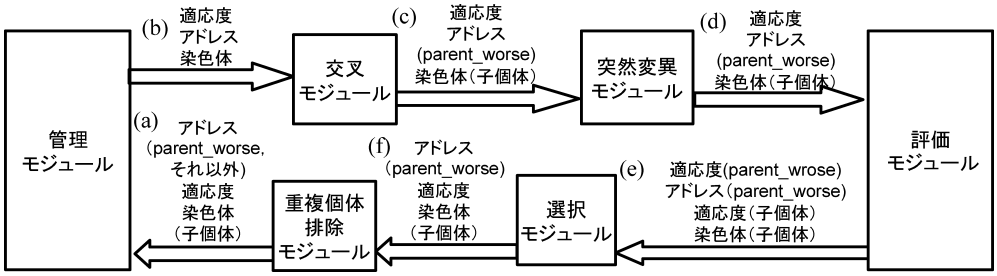

図 3 基本アーキテクチャ（MOGA）
Fig. 3 Basic architecture.

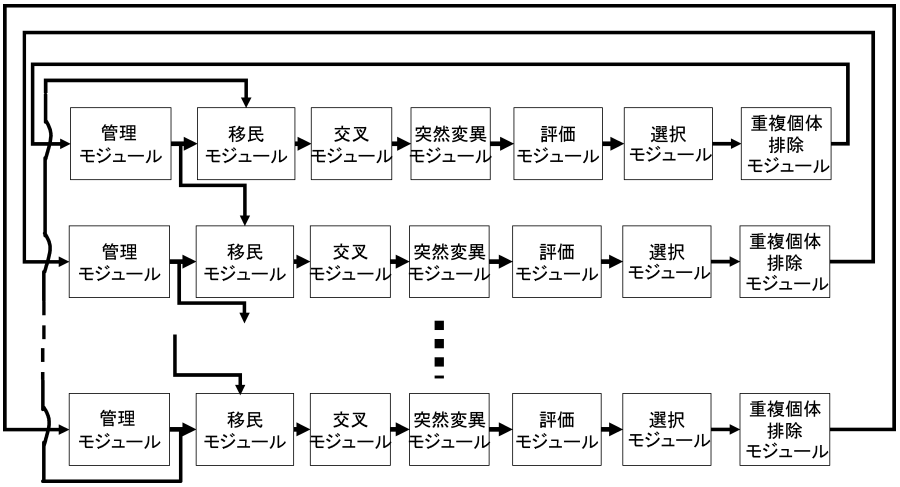

図 4 並列アーキテクチャ（MOGA）
Fig. 4 Parallel architecture.

常選択と偏向選択は MGG モデルによる個体の置き換えを決定するときに用いる個体の優劣を決定する基準が異なる．通常選択は，個体を更新する基準として，子個体が親個体を支配しているとき個体を入れ替える．それに対し，偏向選択では，ある特定の目的に関して，子個体が親個体より優れている場合，子が親を支配していなくても，個体の入れ替えを行う．偏向選択を用いることで特定の目的を優先した探索を行うことができる．また，島間で情報の共有を行うために一定期間ごとに隣りの島から 1 つの個体を移民する．

4. 提案アーキテクチャのハードウェア化

本章では，3 章で述べたアーキテクチャの実装について述べる．提案アーキテクチャは文献 9) で提案されているアーキテクチャを拡張したものであり，モジュール単位で構成されている．提案アーキテクチャで新たに追加したモジュールは，**選択モジュール（selection module）**，**重複個体排除モジュール（overlap rejection module）**である．

提案アーキテクチャは，最小の MOGA 回路を構築するための基本アーキテクチャ（**図 3**）と並列回路を構築するための並列アーキテクチャ（**図 4**）からなる．

モジュールの設計の概要を 4.1 節で述べる．そして従来から存在するモジュール群を 4.2 節で，新たに多目的最適化問題に適用するために追加した拡張モジュール群について 4.3 節で述べる．

4.1 モジュールの設計概要

提案アーキテクチャでは個体は染色体と適応度から構成される．各個体の染色体は，n ビットの 2 進数で表される．モジュール間の染色体の送受信に用いられるバス幅は m ビットとする．n，m の値はパラメータとして与える．各モジュールは，原則として各クロックごとに m ビットのデータを受け取り，c クロック後に m ビットの演算結果を出力できるように設計する（ここで c は定数）．また，データ処理中も，各クロックごとに新たなデータを受け取れるよう設計する．すなわち，$n \geq m$ の場合，$\lceil \frac{n}{m} \rceil$ クロックが 1 染色体あたりのスループットとなる．各モジュールは，順番にデータを受信しつつパイプライン的にデータを逐次処理する．

4.2 従来モジュール群

管理モジュール 管理モジュールは個体群を保持するメモリを含む．このモジュールはメモリより個体を読み出し，それらを 1 つずつ交叉モジュールへ送信する（図 3 (b)）．同時に，重複個体排除モジュールから情報を受信する（図 3 (a)）．その際，このモジュールは重複個体排除モジュールから受信した情報に従い個体の更新を行う．

交叉モジュール 交叉モジュールは $\lceil \frac{n}{m} \rceil$ クロック前に受信した個体（*parent1* とする）の染色体，アドレス，適応度を保持するためのレジスタ r を保持している．交叉モジュールは交叉操作を *parent1* の染色体と受信した最新の個体（*parent2* とする）の染色体に適用し，新しい子個体の染色体を生成し，突然変異モジュールへ送信する．同時に交叉モジュールでは，選択操作の一部も行う．*parent1* と *parent2* の適応度を比べ，支配されている方のアドレス（*parent_worse* とする）と適応度を突然変異モジュールへ出力する（図 3 (c)）．もし，2 つの親が互いに支配しない場合は，*parent1* の個体の適応度とアドレスを交叉モジュールへ送信する．

突然変異モジュール 突然変異モジュールは，交叉モジュールより受信した子の染色体に，与えられた確率で突然変異を適用することによってできた染色体を評価モジュールへ送信する．また，交叉モジュールより受信した親個体のアドレスおよび適応度を評価モジュールへ出力する（図 3 (d)）．

評価モジュール 評価モジュールでは，突然変異モジュールより受信した子の染色体を評価し，各目的に対する適応度を求める．そして，評価モジュールより受信した情報に子の適応度を加えた情報を選択モジュールへ送信する（図 3 (e)）．

移民モジュール 移民モジュールは，回路の並列化を行う際に用いられる．移民モジュールは，通常属している島の管理モジュールと他の島に属する管理モジュールから情報を受信する．通常，交叉モジュールは自身が属する島の管理モジュールから個体を読み込むが，決められた周期で他方の島の管理モジュールから個体を読み込み交叉モジュールへ出力する．

4.3 拡張モジュール群

選択モジュール 選択モジュールは，新たに作成された子個体の適応度と染色体，一方の親個体の適応度，アドレスが入力として与えられる．選択モジュールでは入力された親個体と子個体の適応度を比較し，個体群の中で，子個体を親個体と入れ替えるかどうかを決定する．その結果は重複個体排除モジュールへ *selected_flag* として与えられる（入れ替えが発生するとき，$selected\_flag = 1$ とする）．選択モジュールは子個体の適応度と染色体，*selected flag*，親個体のアドレスを重複個体排除モジュールへ送信する．

重複個体排除モジュール 重複個体排除モジュールは，3.2.2 項で述べた重複個体排除機構を実装する．このモジュールは，選択モジュールより子個体の情報，親個体のアドレス，*selected_flag* を受信する．そして，3.2.2 項で述べた処理を行い，その結果である子個体の情報，アドレス，個体の更新を行うかどうかを表す *overwrite_flag* を管理モジュールへ送信する（個体の更新を行うとき，$overwrite\_flag = true$ とする）．

5. 実験結果・評価

提案アーキテクチャによる MOGA 回路の並列化に対する拡張性，解の探索能力，既存手法に対する優位性を調査するため，いくつかの実験を行った．

5.1 ベンチマーク問題

提案手法を評価するためのベンチマーク問題として，以下の 2 つの問題を用いた．本節では問題の詳細および，それぞれの問題を解くために用いた遺伝演算子について述べる．

多目的ナップサック問題

多目的ナップサック問題として 2KP100-50 [18] を用いた．これは荷物数 100，目的数 2，制約条件 1 を持つ．それぞれの荷物は 2 種類の価値と 1 つの重さを持ち，ナップサックに詰めることのできる重さの合計は制約条件として与えられている．目的関数およびおよび制約条件は以下の式で与えられる．

$$\text{maximize} \sum_{i=1}^{n} c_i^j \cdot x_i$$

$$\text{restriction} \sum_{i=1}^{n} w_i \cdot x_i \leq b$$

$$\text{where } x_i = 0 \text{ or } 1$$

ここで n は荷物の数，c_i^j は荷物 i の目的 j に対する価値，w_i は荷物の重さ，b は重さの合計に対する制約条件である．

このとき，それぞれの価値関数値を最大化するような詰め方の折衷案 x の集合を求めることが目的となる．この 2KP100-50 は 149 のパレート最適解を持ち，実験のために利用することができる．

実験では遺伝演算子として，交叉操作には半一様交叉を用いた．半一様交叉は，染色体上の各遺伝子をどちらの親から引き継ぐかをランダムに決定する手法であり，それぞれの親から引き継ぐ遺伝子の数は同じで

ある．突然変異はビット単位でのランダムな反転を用いた．

多目的巡回セールスマン問題

単一目的の巡回セールスマン問題（以下，TSP）は，都市間を移動するためのコスト（たとえば距離）が与えられたとき，総移動コストを最小にするようなすべての都市を巡回する経路を求める問題である．多目的 TSP では，このコストが複数与えられ（たとえば距離・所要時間），これらコストを最小化する折衷案の集合を求めることが目的となる．先行研究[9] のモジュールを利用するため，本実験では 51 都市の巡回セールスマン問題を用いた．ここではそれぞれの都市に 0 から 99 までの x，y 座標を 2 セットずつランダムに与え，2 都市のユークリッド距離をコストとした．すなわち，都市は 2 つの平面上にそれぞれランダムな座標を与えられる．この問題は元来最小化問題であるため，定数 3,000 から総コストを引いたものをこの問題の適応度とすることにより最大化問題に変換した．この数値は，ほとんどの個体の適応度が負の値にならないような値として定めた．たとえば，それぞれの平面での総移動コストが (800, 1,400) であったとき，適応度は (2,200, 1,600) となる．

実験では，遺伝演算子として，交叉には PMX（Partially Matched Crossover[15]），突然変異はランダムに選ばれた 2 都市間の経路を逆転させる（ひねる）演算子を用いた．

5.2 回路規模と動作周波数

まず，並列アーキテクチャのスケーラビリティを確認するため，ナップサック問題に対して回路の並列度を増加させていったときの，MOGA 回路のサイズと最大動作周波数（Maximum Operating Frequency, MOF）の変化を調べた．MOGA 回路は VHDL で記述し，回路合成には，Quartus II を用いた．このとき，ターゲットデバイスとして，Altera 社の CycloneII シリーズ（回路規模 4,608〜68,416 LE）[19] を用いたが，島あたりの個体数が 64 の場合，実装する回路のサイズが回路規模に収まらず，論理合成ができなかった．提案アーキテクチャは，パイプライン処理での実装を考慮して作成しているため，個体数が増加しても最大動作周波数は低下しないと考え，ここでは島あたりの個体数が 2 で論理合成を行い，それより得られた最大動作周波数を基に計算速度および回路規模を評価した．最大動作周波数と回路規模の一覧を**表 1** に示す．

表 1 の “knapsack” はナップサック問題に対する回路であることを，カッコ内の数字は並列度を示している．表 1 より，並列化による最大動作周波数は，並列度に関係なくほぼ一定となり，本並列アーキテクチャは並列度の増加に対し拡張性があることが分かる．なお，MOGA 回路は島単位で任意に偏向選択と通常選択を切替えできるように設計を行っており，偏向選択を用いた回路は，通常選択を用いた回路と同じ動作周波数で動作が可能である．

表 1 最大動作周波数と回路規模
Table 1 Maximum operating frequency (MOF) and size of circuits.

手法	回路規模 (Logic Elements)	最大動作周波数 (MHz)
knapsack (1)	11,129	62.72
knapsack (2)	23,032	62.45
knapsack (3)	34,475	63.61
knapsack (4)	45,913	63.71
knapsack (6)	67,401	63.78
TSP (1)	3,723	93.98

また TSP に対しても同様の論理合成を行い，回路規模と最大動作周波数を計測した．結果を同表 1 下部，TSP (1) に示す．ナップサック問題に比べて動作周波数が高速であるが，1 個体分の評価処理を行うために複数クロックが必要である．パイプライン処理を考慮したうえで，交叉モジュール（$3d+3$ クロック，ここで d は交叉点間の遺伝子長）と評価モジュール（遺伝子長クロック）の合計値が必要であるため，都市数が 51 であれば，最大 207 クロック，平均 131 クロッククロック数が処理のために必要となる．

5.3 解探索能力

次に，提案アーキテクチャによる MOGA 回路の解探索能力を調査した．本実験では，十分大きなサイズの問題に対して評価を行うため，ソフトウェアで実装したうえで評価を行った．ソフトウェアは，gcc version 4.1.2，最適化オプション O3 を用いてコンパイルを行い，AMD Athlon (tm) 64 X2 Dual Core Processor 4200+，256 MB memory，linux 2.6.18 上で実行した．

評価指標として，支配領域（size of dominated space）を用いた．これは Zitzler による MOGA のサマリ[13] で紹介された手法で，これは得られたパレート最適解によって支配された解空間の面積である．ここでは座標 (0, 0) を起点とした．たとえば評価値 (100, 100) を持つパレート最適解は，評価値の積である 10,000 の面積を支配する．ただし複数のパレート最適解によって重複して支配されている部分領域は 1 度だけこの面積に加算される．

交叉率，突然変異率は，いくつかの値に対して予備実験を行い，最も優れた結果を示したものを採用した．

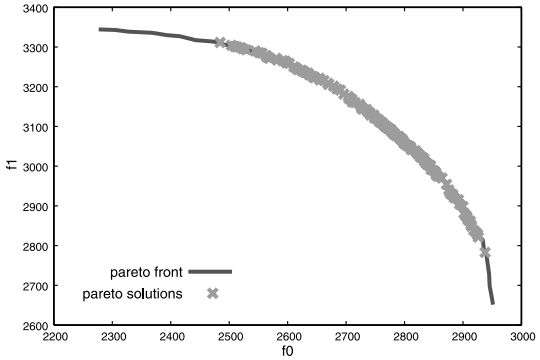

図 5 パレート最適解：normal (1)
Fig. 5 Found solutions: normal (1).

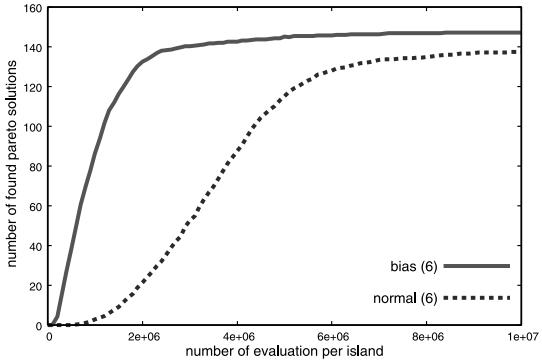

図 6 並列度 6 における得られた多目的最適解の数
Fig. 6 The number of found pareto solutions (6 pipelines).

また，個体数は，予備実験の結果より 128 とした．これは島ごとの個体数であり，島数が 2 であれば総個体数は 256 となる．実験は 10 試行を行い，結果はその平均値を用いた．

5.3.1 多目的ナップサック問題

まず，提案した重複個体排除機構の能力を調査するために，多目的ナップサック問題を用いて実験を行った．十分に解探索が行われたと考えられる 1,000,000 クロック経過時における normal (1) の解空間中の探索状態を**図 5** に示す．図の縦軸および横軸はそれぞれナップサック問題の目的 1，2 に対する適応度を示している．図中の “pareto front” は，解空間中におけるパレートフロントを示し，“non-dominated solutions” は，それぞれの個体群中のパレート最適解を示している．この図では，10 試行を行った実験のうち，最も優れた結果を図示している．また，図を見やすくするために個体群中のパレート最適解のみプロットしている．図 5 より重複個体排除機構を用いることで，パレートフロント上のパレート最適解を多様性を維持しつつ探索していることが分かる．

次に，偏向選択を用いた並列手法の有効性を確認するため，偏向選択を用いた 6 並列の場合（bias (6)），用いなかった場合（normal (6)）の，評価回数あたりのパレート最適解の発見個数を計測した．結果を**図 6** に示す．縦軸は発見されたパレート最適解の数，横軸は島あたりの評価回数を示す．偏向選択を用いた場合，より少ない評価回数で最適解をより多く見つけられていることが分かる．

さらに，提案手法の探索効率を調査するため，80 最適解を求めるまでに必要であった評価回数，ソフトウェア上の処理時間，回路合成時を想定した処理時間，および収束時に得られた最適解の個数と支配領域の面積を計測した．結果を**表 2** に示す．比較対象としては，代表的な MOGA の 1 つである NSGA-II を用いている．NSGA-II は，ソフトウェアで実装されることを前提として設計された MOGA であり，回路化が困難である．そのため，NSGA-II はソフトウェアで実装したものを比較実験に用いた．また，予備実験より NSGA-II の適切なパラメータ値として個体数は 128 と 256，交叉率は 1.0，突然変異率は 0.04 を用いた．個体数を n としたとき，提案手法の 1 個体あたりの計算量は $O(n)$ であり，1 世代あたりの 1 個体しか処理しない．一方，NSGA-II は 1 世代あたり n 個体を処理し，計算量は世代あたり $O(n^2)$ である．これらのことよりソフトウェア上に実装した提案手法と NSGA-II は，処理時間および評価回数を基準に比較を行う．

表 2 の，それぞれの手法で得られたパレート最適解の数が約 80 個に達するまでに要した評価回数と処理時間に着目する．評価時間は，提案アーキテクチャを回路合成した場合を想定したもの，およびソフトウェア上で計算したときのものを用いている．括弧内の数値は並列度である．提案アーキテクチャによる MOGA 回路は並列度が上がるにつれ最終的に得られたパレート最適解の数が向上していることから，提案手法における並列化の効果は大きいことが分かる．また回路換算による提案手法の処理時間を NSGA-II と比較したところ，提案手法は NSGA-II より 1,544〜6,103 倍優れた性能を示している．このことから，提案アーキテクチャを利用して回路合成することの効果は大きいことが分かる．

次に，表 2 の右側に示される支配領域に着目する．normal の提案手法はこの値が小さく，逆に bias のものは大きい．この値は，パレートフロントの両端点（それぞれの目的関数値の最大値）をどれだけ求められたかによって大きく増減する値であるため，bias モ

表 2 提案手法の比較結果（ナップサック問題）
Table 2 Comparison between proposed methods (knapsack problem).

手法	80 パレート最適解			最終的なパレート最適解	支配領域
	評価回数 全島の合計値	処理時間（sec.） ソフトウェア	処理時間（msec.） 回路換算		
normal (1)	2,020,000	22.6	32.2	126	9,657,373
normal (2)	2,520,000	21.1	20.2	128	9,668,371
normal (3)	3,180,000	32.0	16.7	129	9,659,675
normal (4)	13,520,000	136	53.1	132	9,694,154
normal (6)	22,860,000	226	59.7	137	9,716,088
bias (3)	3,540,000	35.8	18.6	135	9,750,411
bias (4)	3,840,000	38.6	15.1	145	9,758,758
bias (6)	5,940,000	58.5	15.5	148	9,759,487
NSGA-II (128)	2,752,000	92.2	–	108	9,737,238
NSGA-II (256)	2,304,000	284	–	142	9,730,358

表 3 提案手法の比較結果（TSP）
Table 3 Comparison between proposed methods (TSP).

手法	支配領域 4,000,000 時			支配領域 5,000,000 時		
	評価回数 全島の合計値	処理時間		評価回数 全島の合計値	処理時間	
		ソフトウェア（sec.）	回路換算（msec.）		ソフトウェア（sec.）	回路換算（msec.）
normal (2)	264,000	1.39	183	34,202,000	179.6	23,770
normal (3)	336,000	1.44	156	29,823,000	127.7	13,818
normal (4)	384,000	1.64	133	16,632,000	71.0	5,780
normal (6)	468,000	2.01	108	25,866,000	111.1	5,992
NSGA-II (128)	213,196	2.00	–	2,043,917	76.7	–
NSGA-II (256)	153,001	12.7	–	11,659,443	969.2	–

デルはこれら端点の個体をうまく発見できたことを示していると考えられる．逆に，NSGA-II (256) はこの値が小さくなっている．得られた結果を調査したところ，最端点の最適解を何試行か見つけられないことがあり，この値を大きく減らしていたことが分かった．このことからも，端点の個体を保持・探索するための bias 機能は有用であると考えられる．

5.3.2 多目的巡回セールスマン問題

提案手法が多様な問題に適用可能であることを示すため，多目的巡回セールスマン問題を用いて実験を行った．実験環境，および比較対象となる NSGA-II の設定は多目的巡回セールスマン問題の場合と同様のものを用いた．

ハードウェア上での処理時間は，シミュレーションによって得られた動作周波数 93.98 MHz，および 1 個体あたりの平均処理クロック数 131 から，1 つの島で 1 回の評価をするために 1.39 マイクロ秒かかるとして計算している．今回用いた問題はランダムに作成され最適解が不明なため，比較評価には支配領域の値を用いた．実験結果を**表 3** に示す．これは得られた非被支配解集合の支配領域の面積が 4,000,000，および 5,000,000 を超えた段階における所要時間と評価回数を示している．この表に島数 1 のときの結果がないのは，支配領域 4,000,000 を超える結果が安定して得られなかったためである．

提案手法は，回路合成を想定した探索速度は並列度を増すにつれて増加している．このことから，提案手法を用いたハードウェア上への実装が有用であることが分かる．しかし，この問題に対しては処理に必要なクロック数が大きいため，ソフトウェア実装された NSGA-II に対してあまり圧倒的な値は出ていない．このような場合は，パイプライン化をすすめるより単純に並列度を増加させたほうが良いとも考えられる．

5.3.3 閾値を用いた重複個体排除機構

最後に，提案する重複排除機構にニッチ法の概念を導入できるかどうかを確かめるため，前実験で良い結果を出した並列度 (4) の提案手法を用い，閾値を変化させたときの実験を行った．ここでは重複個体排除モジュールにおいて個体どうしを比較した際，目的ごとの適応度の差を計算し，それらを加算したものが閾値を下回っていたときは重複していると見なしている．ただし，これら個体が支配–被支配の関係にあった場合には，この値が閾値以下であっても重複であるとは見なさない．評価指標として前実験と同様に，支配領

表 4 重複個体排除機構における閾値の効果
Table 4 Effect of threshold value on overlap rejection module.

手法	支配領域 4,000,000 時			支配領域 5,000,000 時		
	評価回数	処理時間		評価回数	処理時間	
	全島の合計値	ソフトウェア (sec.)	回路換算 (msec.)	全島の合計値	ソフトウェア (sec.)	回路換算 (msec.)
normal (0)	384,000	1.64	133	16,632,000	71.2	5,780
nitch (2)	372,000	1.59	129	14,592,000	62.5	5,071
nitch (5)	376,000	1.61	131	14,532,000	62.2	5,049
nitch (7)	348,000	1.49	121	14,636,000	62.6	5,086
nitch (10)	376,000	1.60	131	17,632,000	75.5	6,127
nitch (20)	384,009	1.64	133	16,144,000	69.1	5,610
nitch (50)	660,000	2.82	229	30,412,000	130	10,568

域の面積が 4,000,000 および 5,000,000 を超えた段階における所要時間と評価回数を用いた．

実験結果を**表 4** に示す．実験の結果，閾値を 2 から 7 あたりに調整すると，探索効率が良くなることが分かった．また逆に，閾値を 50 以上にすると探索効率が悪化することが分かった．この実験に関しては，次章において考察する．

6. 考　　察

本章では，重複個体排除機構の閾値，および提案手法を開発する要因となった NSGA-II のパイプライン実装の難しさに関して考察する．

6.1 重複個体排除機構の閾値

本節では，閾値の適切な値について考察する．ニッチ法では，パレートフロント上にまんべんなく解が得られることが望ましいため，パレートフロントの解空間上の長さと個体数，およびニッチ半径（閾値）の関係が重要である．この問題において，支配領域が 5,000,000 であるとき，得られたパレートフロントの両端の個体はそれぞれ (2,200, 800)，(800, 2,200) 程度の適応度の組を得ていた．よって，マンハッタン距離で考えた場合，パレートフロントの長さは (2,200 − 800) + (2,200 − 800) = 2,800 であると考えられる．この間を 128 個体（島ごとの個体数）で割り振れば 1 個体あたり 21.9 となる．閾値がこの値を超すと，性能は悪化する傾向にあった．実験の結果，51 都市 TSP に対して閾値が 2 から 7 の間であるときに良い結果を示していることから，両端の否被支配個体の最大–最小の適応度の差の合計値を個体数で割り，その値の 1/2 から 1/3 程度の閾値を与えればよいのではないかと考えられる．この値はアルゴリズムの動作中に得られるため，動的に調整するシステムを開発することが今後の課題として考えられる．

6.2 NSGA-II のパイプライン実装の困難性

NSGA-II では，それぞれの個体がどれだけパレート最適に近いかを示すためのランク付けを行う．この際，まずすべての個体どうしを比較し，それぞれの個体について「その個体が支配されている個体の数」を数え，「その個体が支配しているすべての個体」のリストを作成する．その後，「支配されていない個体」にランクを付け，個体群から取り除き，支配関係のリストを更新する．この過程はすべての個体が取り除かれるまで行われる．最後に取り除かれた個体をすべて集め，ソートし，多様性を評価して再びソートする．このソート結果を基に次世代の個体群を決定する．

以上 NSGA-II のアルゴリズムを概説したが，これら過程のほとんどの部分は前段階の作業の完了を前提とするため，パイプライン化することがきわめて難しい．また支配関係のリストを保持するために，個体数 n に対して最悪 n^2 のメモリを必要とするため，モジュール内にメモリをそれぞれ搭載することも非現実的である．以上のことから，NSGA-II は単一のメモリに対し，様々な処理のためのモジュールがアクセスできる形式が適している．つまり，高速な CPU を利用できるマシンを用いてのソフトウェア実装に非常に適していると考えられる．

しかし，NSGA-II の処理は主に比較と個体群のソートであるため，これらを高速に行うことができるならばハードウェア実装に適している可能性がある．今回取り扱った TSP は，1 個体を処理するためには，パイプライン化してさえ平均 138 クロックを必要とした．比較が 1 クロックで済むならば，1 個体に対して全個体（たとえば 128 個体）に対して比較しても十分に許容範囲であると考えられる．またある程度低速であっても，NSGA-II の優れた優良個体の保持機能と，提案手法のような高速化された解探索機能を組み合わせることによって，新たなハイブリッド手法が開発できると考えられる．

7. ま と め

本論文では，MOGA 回路を実装するためのアーキテクチャを提案した．提案アーキテクチャでは，ハードウェア化に適した重複個体を排除するための選択操作と島ごとに異なる部分のパレート最適解を探索するための並列アーキテクチャを含む．提案アーキテクチャによる MOGA 回路の性能を調査するため，提案アーキテクチャによる MOGA と既存の手法である NSGA-II とを比較した．その結果，提案手法は NSGA-II より最大 6,103 倍優れた性能を示すことを確認した．

提案手法では，レジスタ単位でのパイプライン化による高速化に焦点を当てたため，内部レジスタのみを利用し，外部メモリを利用しなかった．今後の課題として，外部メモリの利用による回路規模の縮小，またモジュール単位のパイプライン化による動作周波数の高速化があげられる．

参 考 文 献

1) 廣安知之，廣安博之，三木光範，渡邊真也，上浦二郎：現象論モデルと遺伝的アルゴリズムによるディーゼルエンジン燃焼効率の多目的最適化，自動車技術会論文誌，Vol.35, No.1, pp.51–56 (2004).
2) 大林 茂：航空機の多目的最適設計，人工知能学会誌，Vol.18, No.5, pp.495–510 (2003).
3) 清田高徳，辻 康孝，野田 理，近藤英二：遺伝的アルゴリズムを用いた多目的ファジィ満足化手法による 1 型サーボ系の一設計法，日本機械学会論文集 C 編，Vol.70, No.696, pp.2392–2398 (2004).
4) 若林真一，小出哲士，八田浩一，中山喜勝，後藤睦明，利根直佳：交差手法の適応的選択機能を組み込んだ遺伝的アルゴリズムの LSI チップによる実現，情報処理学会論文誌，Vol.41, No.6, pp. 1766–1776 (2000).
5) 小林亮一，阿部正英，川又政征：遺伝的アルゴリズムを用いた不連続閉曲線抽出手法の FPGA 上での実現，電子情報通信学会技術研究報告，No.CAS2000-131，DSP2000-189，CS2000-151, pp.29–36 (2001).
6) 北浦 理，浅田英昭，松崎元昭，河合隆光，安藤秀樹，島田俊夫：パイプラインストールを除去した遺伝的アルゴリズム専用ハードウェア，計測自動制御学会論文集，Vol.35, No.11, pp.1496–1504 (1999).
7) Shackleford, B., Okushi, E., Yasuda, M., Koizumi, H., Seo, K., Iwamoto, T. and Yasuura, H.: High-performance hardware design and implementation of genetic algorithms, *Hardware Implementation of Intelligent Systems*, Physica-Verlag GmbH, pp.53–87 (2001).
8) Aporntewan, C. and Chongstitvatana, P.: A Hardware Implementation of the Compact Genetic Algorithm, *Proc. 2001 Congress on Evolutionary Computation* (*CEC2001*), pp. 624–629 (2001).
9) 橘 達弘，村田佳洋，柴田直樹，安本慶一，伊藤実：FPGA 上への遺伝的アルゴリズムの柔軟な実装手法の提案，電子情報通信学会論文誌，Vol.J89-D, No.6, pp.1182–1191 (2006).
10) Shimodaira, H.: An Empirical Performance Comparison of Niching Methods for Genetic Algorithms, *IEICE Trans. Inf. & Syst.*, Vol.E85-D, No.11, pp.1872–1880 (2002).
11) Deb, K., Agrawal, S., Pratab, A. and Meyarivan, T.: A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II, *Proc. Parallel Problem Solving from Nature VI Conference*, pp. 849–858 (2000).
12) Zitzler, E., Laumanns, M. and Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm, Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland (2001).
13) Zitzler, E.: Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications, Ph.D. thesis, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland (1999).
14) Horn, J., Nafpliotis, N. and Goldberg, D.E.: A niched Pareto genetic algorithm for multiobjective optimization, *Proc. IEEE Symposium on Circuits and Systems*, pp.2264–2267 (1991).
15) Goldberg, D.E. and Lingle, R.: Alleles, loci and the traveling salesman problem, *Proc. Int'l Conf. on Genetic Algorithms and their Applications*, pp.154–159 (1985).
16) Satoh, H., Ono, I. and Kobayashi, S.: Minimal Generation Gap Model for GAs Considering Both Exploration and Exploitation, *Proc. 4th Int'l Conf. on Soft Computing* (*IIZUKA'96*), pp.494–497 (1996).
17) Chafekar, D., Xuan, J. and Rasheed, K.: Constrained Multi-Objective Optimization Using Steady State Genetic Algorithms, *Proc. Genetic and Evolutionary Computation Conference* (*GECCO*), pp.813–824 (2003).
18) Gandibleux, X.: MCDM Numerical Instances Library. http://www.univ-valenciennes.fr/ROAD/MCDM/ListMOKP.html
19) Cyclone II デバイス・ファミリ・データ・シート．http://www.altera.co.jp/literature/hb/cyc2/cyc2_cii5v1_01_j.pdf

(平成 19 年 4 月 27 日受付)
(平成 19 年 10 月 2 日採録)

橘　　達弘（学生会員）

平成 16 年奈良先端科学技術大学院大学情報科学研究科情報処理学専攻博士前期課程修了．平成 19 年奈良先端科学技術大学院大学情報科学研究科情報処理学専攻博士後期課程修了．遺伝的アルゴリズムの研究に従事．

村田　佳洋（正会員）

平成 12 年奈良先端科学技術大学院大学情報科学研究科修士課程修了．平成 15 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．平成 15 年 4 月より奈良先端科学技術大学院大学情報科学研究科助手．遺伝的アルゴリズムとエージェント技術の研究に従事．

柴田　直樹（正会員）

平成 13 年大阪大学大学院基礎工学研究科情報数理系専攻博士後期課程修了．平成 13 年 4 月より奈良先端科学技術大学院大学情報科学研究科助手．平成 16 年 4 月より滋賀大学経済学部情報管理学科准教授．分散システム，遺伝的アルゴリズムと ITS 等の研究に従事．

安本　慶一（正会員）

平成 3 年大阪大学基礎工学部情報工学科卒業．平成 7 年大阪大学大学院博士後期課程退学後，滋賀大学経済学部助手．平成 14 年より現在，奈良先端科学技術大学院大学情報科学研究科准教授．博士（工学）．分散システム，マルチメディア通信システムに関する研究に従事．ACM，IEEE/CS 各会員．

伊藤　　実（正会員）

昭和 52 年大阪大学基礎工学部情報工学科卒業．昭和 54 年大阪大学大学院基礎工学研究科博士後期課程退学後，同大学助手．昭和 58 年工学博士（大阪大学）．平成 3 年カナダウォータールー大学数学部客員準教授．平成 5 年奈良先端科学技術大学院大学情報科学研究科教授．データベース理論，分散システム等に関する研究に従事．電子情報通信学会，ACM，IEEE/CS 各会員．