CATS Communication Art Technology Systems
CESL

遺伝的アルゴリズム GA 入門

～遺伝的アルゴリズムを用いたテストケースの生成～

目時 伸哉†

本稿では最適化アルゴリズムの 1 つである遺伝的アルゴリズム GA の入門的解説を行う. 遺伝的アルゴリズム GA が Michigan 大学の J. H. Holland によって最初に提唱されたのは 1970 年代のことである. それ以来, GA は他の最適化手法に比べ, 比較的早い時間で近似解を探索できると期待されている. 本稿では, まず GA を取り巻く最適化手法全般について説明する. 続いて GA に関する用語の定義や手法を解説し, 最悪実行時間を検証すべくテストケース群の生成に GA を適用した事例を紹介する.

An introduction to Genetic Algorithm

～GA-based Test Case Generation～

SHINYA METOKI†

In this paper, we introduce the fundamental notions of genetic algorithm (GA), which is one of optimization algorithms. Genetic algorithm was formally introduced in the 1970s by J. H. Holland at University of Michigan, and it has been considered to find approximate solutions of the problem in a shorter time than other optimization methods. The first part of this paper is devoted to explanations of general optimization methods surrounding GA .In the next, we describe definition of terms and methods related to GA. Then we introduce a case example in which GA enables us to generate more efficiently test cases to validating worst case execution time of some kind of job shop scheduling problem.

1. はじめに

地球が誕生したのは現在を遡ること約 46 億年前, 最初の生命が誕生したのは約 36 億年前と云われている. その間, 生命の連鎖は綿々と受け継がれてきた. 恐竜のように絶滅した種もあれば, 古細菌のように最初の生命として誕生して以来, 姿・形を変えることなく現在も生き延びている種もある. しかし, 我々ヒトを含め多くの種は, 悠久の時間を経て徐々に進化し, 現在の姿に到っている. 生物界の進化・淘汰を考える上で, 鍵となるのが遺伝子 DNA である. 遺伝的アルゴリズム GA(genetic algorithm)とは, コンピュータ上で扱う諸々のデータを遺伝子 DNA として捉え, そこに偶然の要素を加えながら, 最適解を発見・探索するシミュレーション技法である.

†キャッツ組込みソフトウェア研究所, CATS Embedded Software Laboratory

近年のソフトウェアの開発規模は巨大化の一途を辿り, それを扱うシステムやネットーワークもまた複雑化する一方である. 設計されたソフトウェアに関して, 全てのテストケースを網羅的に検証することは, 事実上不可能になっている. これに対し, 従来であれば, ランダムにテストケースを生成し, 検証する手法が用いられてきたが, 決して効率がよく, 信頼のおける手法とは言えない. 一方, 遺伝的アルゴリズム GA を用いて, ソフトウェアシステムに対して, 比較的負荷の高かったテストケース群からさらに負荷の高いと推定されるテストケース群を次々と生成することで, 効率のよい検証を実施することが期待できるだろう.

本稿の構成は以下のようになっている.

まず, 第 2 節で遺伝的アルゴリズム GA を取り巻く, 一般の最適化アルゴリズムを紹介し, 遺伝的アルゴリズム GA の位置づけを確認する.

第 3 節で遺伝的アルゴリズム GA の用語・手法を説

明し，また GA が抱える問題点についても考察する．

第 4 節では，スケジューリング問題 JSSP の最悪実行時間 WCET を検証すべくテストケース群の生成に GA を適用した事例を紹介する．

2. 最適化アルゴリズム～最適解を探索するアルゴリズム～

応用数学において「ある制約条件の下で，ある関数の値を最大化または最小化する解を探すアルゴリズム」を広く一般的に「最適化アルゴリズム」と呼んでいる．ここで問題の主体となっている「ある関数」を「目的関数」と言う．

最適化アルゴリズムは，線型計画法 LP・非線型計画法 NP・動的計画法 DP そして進化的アルゴリズム EA から成る．

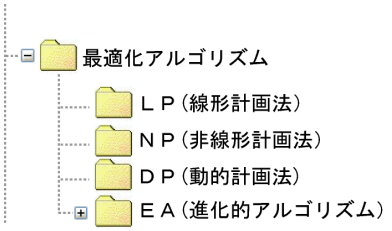

図 1 最適化アルゴリズムの分類

2.1. 線型計画法 Linear Programming

制約条件と目的関数がともに 1 次式で表される，最も単純な最適化アルゴリズムである．

例えば，2 種類の製品 X と Y があるとしよう．それらを製造するために 2 つの原料 a と b があり，X を 1 つ製造するために a を 16kg，b を 20kg だけ使い，同じく Y を 1 つ製造するために a を 25kg，b を 10kg だけ使う．しかし，a と b の残り在庫量はそれぞれ 250kg と 180kg である．X と Y それぞれ 1 つあたりの利益が 3 万円および 2 万円とする．限られた残りの在庫量の中で，利益を最大にするためには，X と Y をそれぞれ何個ずつ製造すればよいか？

表 1 線型計画法

	X への使用量	Y への使用量	在庫量
原料 a	16 kg	25 kg	250kg
原料 b	20 kg	10 kg	180kg

製品 X と Y をそれぞれ x 個および y 個製造するとしよう．このとき原料 a の使用量は(16x+25y)kg，原料 b の使用量は(20x+10y)kg になる．これらは残りの在庫量の範囲内でなければならない．これが制約条件である:

$$16x+25y \leqq 250$$
$$20x+10y \leqq 180$$
$$x, y \geqq 0$$

一方，利益は(3x+2y)万円となり，これを最大にしたい．これが目的関数である:

$$f(x, y) = 3x+2y$$

制約条件と目的関数がともに 1 次式で表されているので，線型計画法の適用範囲である．最適解の求め方は単純で，グラフを描くことによって求まる:

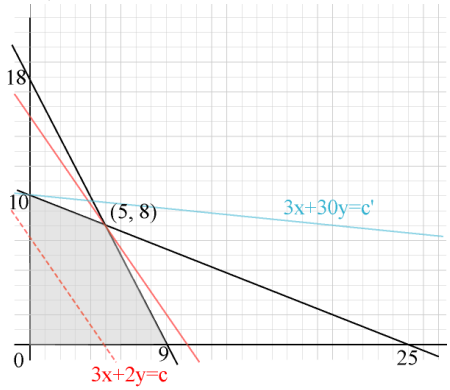

図 2 線型計画表

灰色に塗ってある領域が制約条件である．利益を c 万円とおくと 3x+2y=c となり，これを図示すると赤い点線で示した直線になり，c の値を大きくすると赤い点線は右上に平行移動する．制約条件すなわち灰色の領域と共有点を持ちながら，利益 c 万円を最大にするのは，点(5,8)を通る赤い実線のときである．よって，X と Y をそれぞれ 5 個と 8 個製造すればよい．ちなみに問題を改変して，Y1 つの利益を 30 万円に替えてみる．この場合，青い直線のように点(0,10)を通るとき，利益 c'万円が最大になる．これは X を一切製造せず，原料の許す範囲で Y だけを 10 個製造することを意味する．X に比べて，Y の利益が極端に大きいことを省みれば，納得できる結果であろう．

2.2. 非線型計画法 Nonlinear Programming

線型計画法 LP と問題の枠組みは同じであるが，制約条件もしくは目的関数に非線型な（1 次式で表せない）項が現れる場合を扱う．問題の種類・性質に応じて，様々なバリエーションが考えられる:

- ラグランジュの未定乗数法
- 勾配法

・Newton 法 など

これらは微分可能性を前提とした解析的な手法である.

2.3. 動的計画法 Dynamic Programming

動的計画法では,「最初の状態と最初の決定がどのようなものであっても,残りの決定は最初の決定から生じた状態に対して最適なものでなければならない」と云う最適化の原理を用いる.これは「最適経路中の部分経路もまた最適経路になっている」とも言い換えられる.

大きな問題の解を小さな問題の解の系列に帰着できる場合に有効である.小さな問題に分割していくと,同じ問題が何度も現れる場合がある.その度ごとに解くのではなく,その解を保存しておけば,必要なときに適宜参照することで効率良く問題を解くことができる.事実,多項式時間では解けないと予想される一部の問題に対しても,動的計画法によって,擬似多項式時間で解くができる.

例えば,次のようにどれも長さの違う 8 本のロウソクが点灯している.いくつかのロウソクの火を消して,その結果,左から右に向かって,火が右肩上がりに見えるようにする.点灯しているロウソクの数を最大にしたい場合,どのロウソクを消せばよいか?

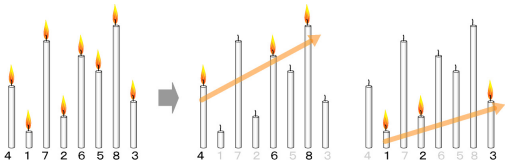

図 3 増加部分列の中で最長のものは?

全ての組合せを網羅しようとするならば,各 8 本のロウソクの点灯・消灯を考えるので 2^8=128 通りある.一方,動的計画法を用いると,図 4のように最長部分列の長さを求めることができる.

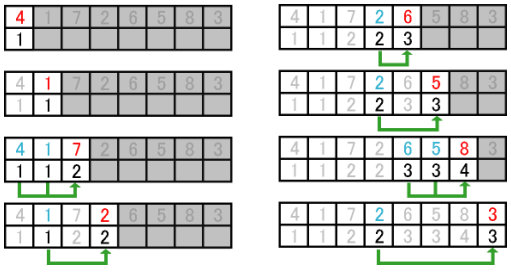

図 4 動的計画法と最長部分増加列

(1)まず,左から順にロウソクの長さ(赤い数字)に着目し,そのロウソクを末尾とする最長部分列の長さを下の段に書く.
(2)具体的には,着目しているロウソクより左側にあり,かつ長さの短いロウソクの存在を調べる.
(3a)存在した場合,それらのうちで,下段の数が最大のロウソク(青い数字)を選ぶ.その下段の数+1 を着目しているロウソクの下段に書く.(緑の矢印)
(3b)存在しない場合,単に下段に 1 と書く.
(4)下段の数の最大値が,最長部分増加列の長さである.

最長部分増加列自体は,最大値の上段の数から始めて,緑の矢印を逆に遡ればよい.図 4の場合,最大値 4 の上段の 8 へ向かう矢印を逆に辿っていくと,

1→2→(6 か 5)→8

が最長部分増加列である.

ロウソクが n 本の場合,網羅探索するには 2^n 通りの指数関数的オーダーになるが,この動的計画法を用いことで, n^2 の多項式オーダーで行うことができる.

2.4. 進化的アルゴリズム Evolutionary Algorithm

EA は,その方法論によって,GA・GP・ES・EP の 4 つに大きく区分される.そのいずれもが,自然界における交叉・突然変異・淘汰・適者生存といった進化の仕組みをモチーフにして,最適化問題の解を求めるアルゴリズムである.

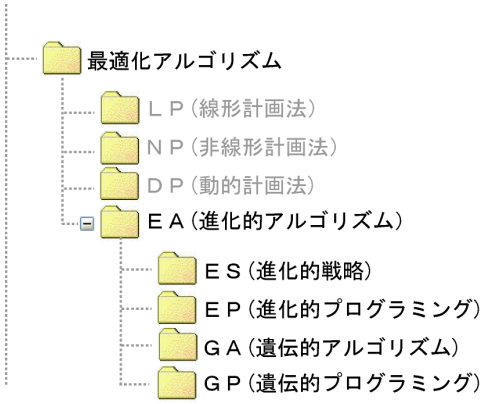

図 5 進化的アルゴリズム EA の分類

EA に属するこれらのアルゴリズムは総じて「メタヒューリスティク(meta-heuristic)」と呼ばれることが多い.ここで「ヒューリスティク(heuristic)」とは,必ずしも厳密解(完全に正しい解)が得られるわけではないが,ある程度までなら厳密解に近い解を得られる方法を意味する.精度は保証されないが,比較的短時間で解を得ること

ができる．一方，「メタ(meta)」とは，特定の問題に特化することなく，広範な問題に対応できることを意味する．

特に GA においては，探索空間の構造や目的関数の微分可能性を必要としない．そのため，線型計画法や非線型計画法が適用できない分野の問題にも対応できる．

3. 遺伝的アルゴリズム GA～ダーウィンの進化論に基づく最適解探索手法～

遺伝的アルゴリズム GA が最初に提唱されたのは，1975 年，ミシガン大学の J. H. Holland の著書“Adaptation in Natural and Artificial Systems”においてである．その後，人工知能隆盛の時期に影を潜めたが，ニューラルネットワークの流行に伴い再び注目を集めるようになった．この節では，遺伝的アルゴリズム GA で用いる用語や手法を解説する．

3.1. 用語定義

まず GA で用いる用語を整理しておこう:

染色体:何かしらの情報を持った個体

遺伝子:染色体に書き込まれた情報の構成要素

遺伝子座:遺伝子が配置される位置

対立遺伝子:各遺伝子座で，遺伝子が取りうる値

遺伝子型:遺伝子による表現(表現型をエンコードしたもの)

表現型:遺伝子型をデコードしたもの

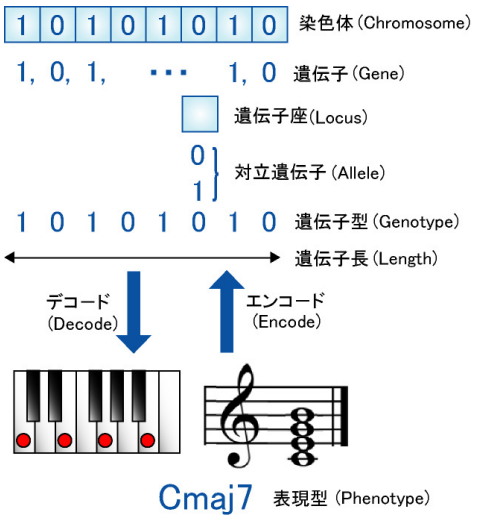

図 6 遺伝的アルゴリズム GA で用いる用語 1

図 6の例においては，和音と言う表現型を 1 オクターブ 8 箇所の鍵盤を押す(1)，押さない(0)で 2 進数に変換し，1 バイトの遺伝子型にエンコードしている．ただし，対立遺伝子はいかなる問題でも 0 か 1 の 2 進表示であるとは限らない．例えば，巡回セールスマン問題 TSP(Traveling Salesman Problem)の順路を遺伝子型で表示する場合，重複のない自然数の順列で表すことが自然であろう．

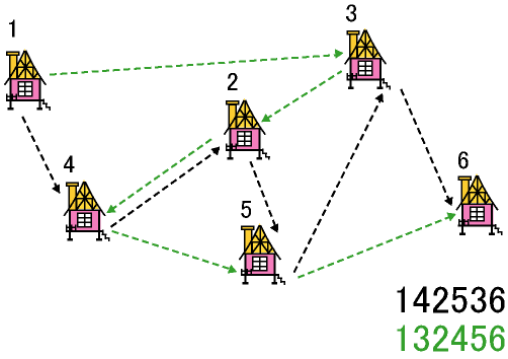

図 7 循環セールスマン問題の遺伝子型

さらに個体と集団を以下のように定義する:

個体: 染色体で特徴付けられた個々

集団: 個体の集まり

集団のサイズ: 集団内の個体数

世代: 遺伝的操作の前または後の集団

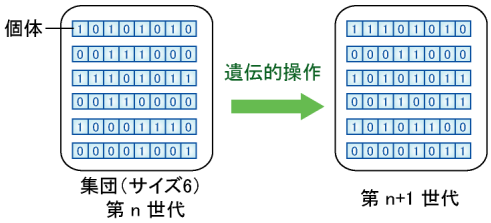

図 8 遺伝的アルゴリズム GA で用いる用語 2

遺伝的操作の内容については，第3.3節で述べる．

3.2. アルゴリズムの手順

遺伝的アルゴリズムGAの手順について，大まかには次のようになっている．

1. 初期集団の生成
2. 適応度の評価
3. 遺伝的操作
4. 終了条件の判定⇒2 に戻る

手順4で終了条件が満たされていれば，そこで終了する．満たされていない場合は,手順2に戻る．

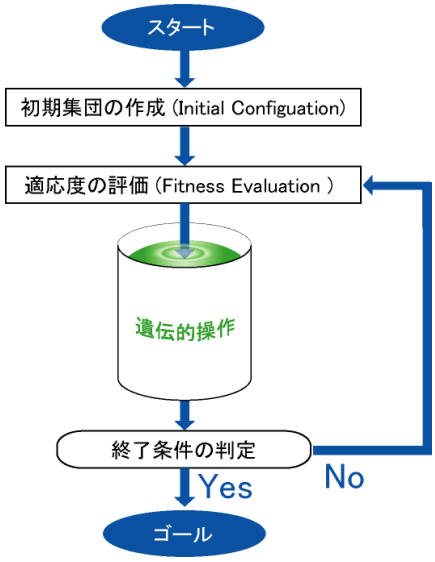

図 9 遺伝的アルゴリズム GA のフローチャート

各手順において，様々なバリエーションがある．以下で詳細に見ていこう．

3.2.1. 初期集団の作成

初期集団はランダムにつくる．ここで気になることは，「集団のサイズをいくらにするか？」と云うことである．

集団のサイズを小さくすると，1世代あたりの計算量は減るが，収束するまでの世代数が大きくなる．逆に集団サイズを大きくすると，収束するまでの世代数は小さくなるが，1世代あたりの計算量が増える．

一般に遺伝子長が長いほど，集団のサイズを大きくとらなければならない．

集団のサイズのようにシミュレーションする側で調節する必要のある値を GA の「パラメータ」という．集団のサイズの他にも，後述の交叉率・突然変異率など様々なパラメータがある．パラメータの設定次第で，シミュレーションの結果も大きく左右されるので，予備実験などを通して慎重に設定する必要がある．

3.2.2. 適応度の評価

適応度とは，集団上の関数である．集団に含まれる各個体について，目的関数の値を加工することで得られる．

目的関数の大小と適応度の高低が一致する場合（利益や得点など）は，そのまま対応させでもよい．逆になる場合（コストや時間など），-1 倍や（正の数であれば）逆数をとるなどの加工が必要である．

3.2.3. 終了条件の判定

図 6 のように，遺伝的アルゴリズム GA では繰り返して処理を行う．何を以って終了の条件とするか？いくつかのバリエーションがある:

a. あらかじめ決めておいた回数分処理を繰り返した

b. あらかじめ決めておいた閾値を超える適応度をもつ個体が現れた

c. あらかじめ決めておいた閾値を超える適応度の平均値をもつ世代(集団)が現れた

d. 世代ごとの適応度の平均値の増加率が，ある閾値以下で，かつある一定期間続いた

終了条件が満たされた場合，アルゴリズムを終了する．最終世代の集団の中で，最も適応度の高い個体を最適解と定める．

a の場合，終了までの時間経過がある程度まで見積もれるが，必ずしも相応しい最適解が得られる保証はない．

b または c の場合，想定していた水準の最適解を得ることができるが，それが真の最適解である保証はない．また適応度の閾値を高く設定しすぎると，繰り返し処理が永久に終わらない恐れがある．

d は世代を経ることによる成長が頭打ちになることを意味している．終了までの時間経過や最適解の水準を事前に見積もることができない．

バリエーションの選択およびパラメータの設定値に関して，扱う問題やシミュレーション環境に応じて適宜使い分ける必要がある．

3.3. 遺伝的操作

選択・交叉・突然変異・淘汰などの生物の進化のプロセスを模倣した操作を現世代の集団に対して適用する．これらの操作の組合せや順序についても，やはり幾つかのバリエーションが存在するが，ここでは「適応度の高い個体がより多くの子孫を残す」原理に基づく，代表的なものについて説明する．

まず現世代の集団から始めて，適応度に応じて親となる個体を「選択」する．2 つの選択した個体を「交叉」し，新たな 2 つの個体の遺伝子をつくる．それらの遺伝子に「突然変異」を反映させて，新たな個体を確定する．

この一連の操作を繰り返し，次世代の個体数が集団サイズに達したら，終了とする．

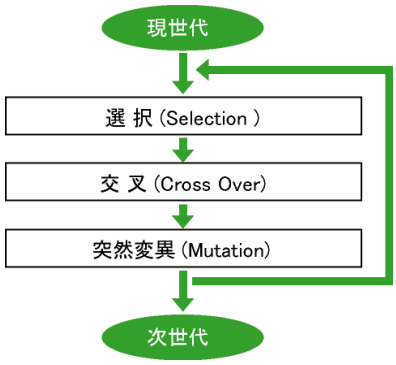

図 10 遺伝的操作のバリエーション 1

ここでは，「淘汰」という言葉が現れていないが，選択の際，適応度の低い個体は親として選択される確率が低くなっている．現世代の個体は，（後述の3.3.1.4エリート選択を除いて）次世代に生き残らない．つまり，適応度の低い個体は親として子孫を残せない意味で自然に淘汰されることになる．

一方，積極的に淘汰の操作を行うバリエーションもある．この場合，現世代から親となる個体を（選択ではなく）ランダムに選び，交叉させる．またランダムに選び，突然変異させる．交叉をm回，突然変異をn回行った結果，サイズ 2m+n の新しい個体の集団がつくられる．この新しい個体の集団と現世代を合併した集団をつくる．現世代のサイズを N とすれば，この合併集団のサイズは N+2m+n になる．この合併集団の各個体の適応度を測り，上位 N 位までの個体を以って次世代として選択し，それ以外の個体を淘汰する．

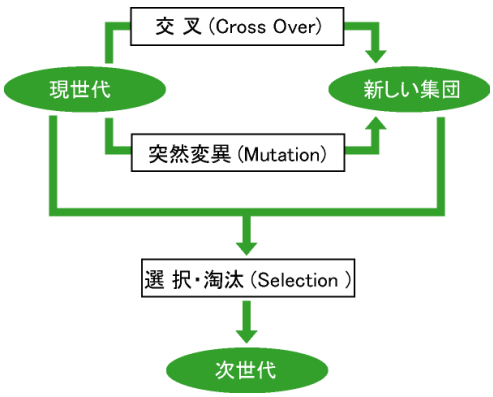

図 11 遺伝的操作のバリエーション 2

3.3.1. 選択

次世代の個体をつくる両親となる 2 つの個体を選ぶ．代表的な選択としては 4 つある:

a. ルーレット選択
b. ランキング選択
c. トーナメント選択
d. エリート選択

3.3.1.1. ルーレット選択

遺伝的アルゴリズムの提唱者 J. H. Holland が提案した選択法である．

まず

G(k)={x_1, x_2, ...x_N}: 第 k 世代の集団
f_i: 個体 x_i の適応度

とするとき，個体 x_j が選択される確率 p_j を

p_j= f_j / (f_1+ f_2+ … + f_N)

で定める．すなわち，各個体が，その適応度に比例した確率で選択されることになる．

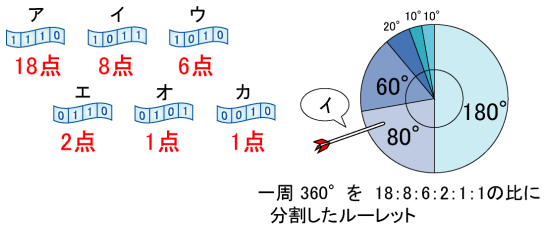

図 12 ルーレット選択

この選択法はシンプルであるが，適応度が負の値をとる場合，このままでは使えない．

3.3.1.2. ランキング選択

適応度の値ではなく，順位に基づいて，各個体を選択する確率を決める．すなわち「適応度 1 位なら確率 q_1， 2 位なら確率 q_2…」と定めておく．

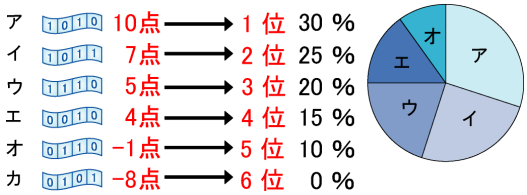

図 13 ランキング選択

適応度が負の値をとる場合でも採用できるが，新しい世代の評価の度にソートする手間がかかる．

3.3.1.3. トーナメント選択

毎回，一定の数だけランダムに個体を取り出し，その中で最も適応度の高い個体を選択する．ここで「一定の数」のことを「トーナメントサイズ」という．

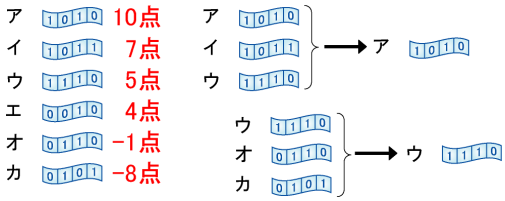

図 14 トーナメント選択

トーナメントサイズを大きくしすぎると適応度の高い同じ個体ばかりが選択される恐れがある．逆に小さくしすぎると，適応度の高い個体でもトーナメントに参加できなかった場合，選択されない恐れもある．またトーナメントサイズを M とすると，最下位から数えて M-1 番目までの個体が決して選択されることはない．

3.3.1.4. エリート選択

適応度の高い順に，決まった数の個体を次世代に移す．既述の 3 つの選択では，選択した後に交叉を行うが，エリート選択では，交叉を行わない．

現世代で適応度の高い個体の遺伝子型を，交叉や突然変異で壊すことなく，次世代に残す目的で行う．

エリート選択のみが単独で採用させることはなく，通常は他の選択法と組み合せて行う．

3.3.2. 交叉

現世代から 2 つの個体を選択した後，それらを交叉し，次世代の 2 つの個体の遺伝子をつくる．代表的な交叉としては 4 つある:

a. 1 点交叉
b. 2 点交叉
c. 多点交叉
d. 一様交叉

3.3.2.1. 1 点交叉

J. H. Holland が提案した最も単純な交叉法である．遺伝子長を L とするとき，(L-1)箇所ある間隙から交叉する 1 点を選び，その前後で遺伝子を入れ替える．

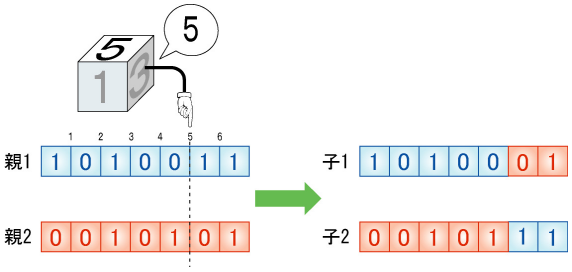

図 15 1 点交叉

実際には効率が悪く，あまり用いられることはない．

3.3.2.2. 2 点交叉

遺伝子長を L とするとき，(L-1)箇所ある間隙から交叉する 2 点を選び，その前後で遺伝子を入れ替える．

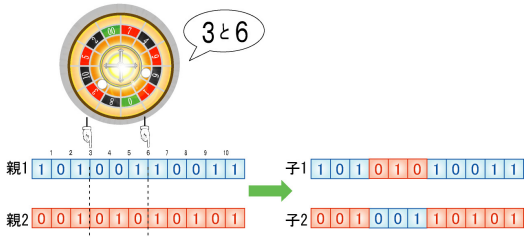

図 16 2 点交叉

3.3.2.3. 多点交叉

2 点交叉の拡張で，間隙から交叉する 3 点以上を選ぶ．しかし，2 点交叉以上の効果は得られないので，あまり用いられることはない．

3.3.2.4. 一様交叉

遺伝子長と同じランダムなビット列をつくり，各位置について，0 ならばその位置の遺伝子を入れ替えず，1 ならばその位置の遺伝子を入れ替える．

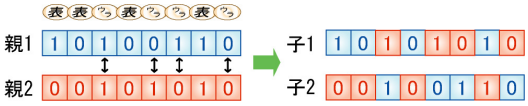

図 17 一様交叉

3.3.3. 突然変位

交叉によって次世代の個体の遺伝子をつくったが，このままでは両親の性質を混合・分離しただけで，そ

の範囲を超えた個体は生まれない．交叉の結果生じた遺伝子に一定の確率で突然変異を施す必要がある．突然変異には，集団を形成する個体の多様性を維持する意図がある．代表的な突然変異の例を挙げる:

a. 単一遺伝子座突然変異
b. 逆位
c. 転座

3.3.3.1. 単一遺伝子座突然変異

ある遺伝子座の 1 箇所を他の対立遺伝子に置き換える．

3.3.3.2. 単一遺伝子座突然変異

ランダムに選んだ 2 点間の遺伝子の順序を逆にする．

3.3.3.3. 転座

ランダムに選んだ 2 点間の遺伝子を別の位置を変える．

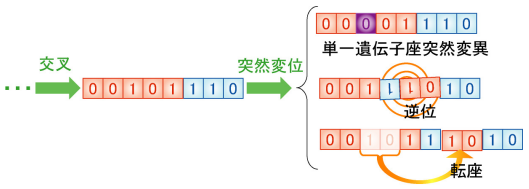

図 18 突然変異

3.4. 遺伝的アルゴリズムの問題点

以上では遺伝的アルゴリズムに関する用語・手順・手法を見てきた．ここでは，実際にアルゴリズムを実行するに際し，起こりうる典型的な問題を挙げる．

a. 初期収束
b. ヒッチハイキング
c. ビットクリフ
d. GA 困難

3.4.1. 初期収束

比較的初期に近い世代において，他の個体に比べて著しく適合度の高い個体が存在した場合，その個体の遺伝子が急速に母集団内に広がる．その結果，母集団の多様性が減少し，また局所最適解に収束する可能性が高くなる．

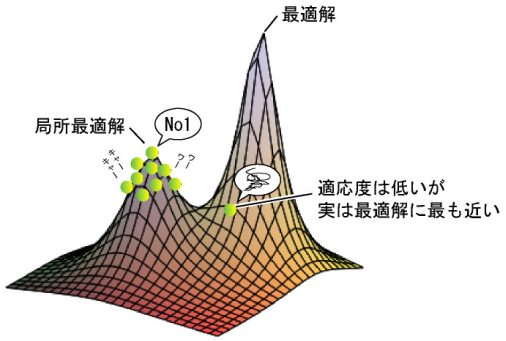

図 19 局所最適解への収束

これを避けるために，近傍に多くの個体が集中する場合，適応度を低く評価するなどのペナルティを課すことで，個体間の棲み分け(sharing)を促し，集団の多様性を図る方策がある．

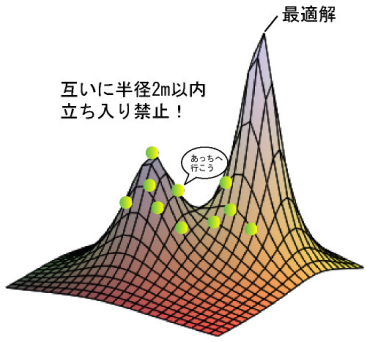

図 20 棲み分け

3.4.2. ヒッチハイク

最適解に極めて近い 2 つの個体でありながら，交叉によって最適解を得にくい現象がある．図 21において，2 点交叉によって，最適解が得られる確率を考えよう．遺伝子長が 11 なので，交叉点(隙間)は 10 点ある．10 点から 2 点を選ぶ組合せの数は全部で 10×9/2=45 通りある．一方，最適解を得るための選び方は，図中に示した 2 通りなので，求める確率は 2/45 およそ 4.4%である．遺伝子長が長くなるに従って，この確率は著しく減少していく．

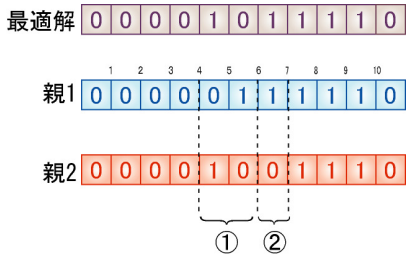

図 21 ヒッチハイカー

これを避けるためには，一様交叉を用いればよい．図 21で最適解を得るための確率は，$1/(2^3)+1/(2^3)=1/4$ すなわち 25%になる．しかも，この確率は遺伝子長が長くなっても同じである．

3.4.3. ビットクリフ

整数や実数の表現型を，2 進数の遺伝子で表現している場合を考えよう．10 進数では近いにも拘らず，2 進表示でハミング距離が遠いことが起こる．例えば，$14=01110_{(2)}$と $15=01111_{(2)}$ならば，ハミング距離は 1 であるが，$15=01111_{(2)}$と $16=10000_{(2)}$では 5 になる．

適応度を評価する関数が連続関数だった場合，15 と16 での値は近いことが期待される．仮に15 が最適解であったとすれば，16 も比較的高い適応度を示す．アルゴリズムの進行中，多くの個体が $16=10000_{(2)}$に集中してきた場合，突然変位を経ても $15=01111_{(2)}$に辿り着く可能性は極めて低くなる．これをビットクリフ(ハミングクリフ)という．

3.4.4. これを避けるために，Gray 表現がある．Gray 表現では隣り合う値のハミング距離は常に 1 になる特徴がある．

10進表現	2進表現	Gray 表現	8	$1000_{(2)}$	$1100_{(g)}$
1	$0001_{(2)}$	$0001_{(g)}$	9	$1001_{(2)}$	$1101_{(g)}$
2	$0010_{(2)}$	$0011_{(g)}$	10	$1010_{(2)}$	$1111_{(g)}$
3	$0011_{(2)}$	$0010_{(g)}$	11	$1011_{(2)}$	$1110_{(g)}$
4	$0100_{(2)}$	$0110_{(g)}$	12	$1100_{(2)}$	$1010_{(g)}$
5	$0101_{(2)}$	$0111_{(g)}$	13	$1101_{(2)}$	$1011_{(g)}$
6	$0110_{(2)}$	$0101_{(g)}$	14	$1110_{(2)}$	$1001_{(g)}$
7	$0111_{(2)}$	$0100_{(g)}$	15	$1111_{(2)}$	$1000_{(g)}$

例）12のGray 表現の求め方

① 12の2進表現を書く $1100_{(2)}$

② 1ビット右にシフトさせる $+\ 1100_{(2)}$

③ 各位で排他的論理和をとる $1010_{(g)}$

図 22 Gray 表現

3.4.5. GA 困難

上の3.4.3のように遺伝子のエンコード法を替えるだけで，解決する問題点もあるが，本質的に困難な問題もある．表 2は有名なビットだまし問題(Ugly bit deceptive functions)である．比較的 0 の多い遺伝子の適応度が高い傾向があるにも拘らず，1111 が最適解になっている．

表 2 ビットだまし問題

遺伝子	1111	0000	0001	0010
適応度	30	28	26	24
遺伝子	0100	1000	0011	0101
適応度	22	20	18	16
遺伝子	0110	1001	1010	1100
適応度	14	12	10	8
遺伝子	1110	1101	1011	0111
適応度	6	4	2	0

GA の解の探索が上手く機能するためには，「最適解に近い良い解は，部分解が組み合わされて生成される」という積み木仮説(building block hypothesis)の成立が必要である．

しかしながら表 2のように，全探索空間または超平面に含まれる低次の超平面(例えば最初の遺伝子が 0 であること)の適応度が大域最適解(1111)または最良超平面の存在方向を指し示さない問題においては，積み木仮説は成立しない．

このような問題を GA 困難という．GA 困難では局所最適解への収束に陥りやすく，従って最適解が発見できないことが多い．

これを避けるためには，これまで述べてきた単純 GA の枠組みを越えた様々な拡張やバリエーションが提案されている．

4. GA の適用事例

リアルタイムシステムの組込みソフトウェアの場合，最悪実行時間 WCET(Worst-case execution time)以内に指定されたスケジュールを全て完了できることが，そのソフトウェアの信頼性に関して重要なことになってくる．

この節では，スケジューリング問題 JSSP(Job Shop Scheduling Problem)に GA を適用することによって，比較的負荷の高かった（時間のかかった）テストケース群

からさらに負荷の高いと推定されるテストケース群を次々と生成することを試みる．これによって，効率のよい検証が期待できる．

4.1. 問題設定(WCET)

3 つの仕事 Job0・Job1・Job2 があり，いずれも 4 つのマシーン M0・M1・M2・M3 による作業を要する．各仕事がマシーンの順番と日数は以下の表の通りである．

表 3 JSSP の問題例

	(仕様マシーン，時間数)			
J0	(M0, 3)	(M1, 3)	(M2, 4)	(M3, 1)
J1	(M3, 1)	(M0, 2)	(M2, 5)	(M1, 2)
J2	(M1, 1)	(M3, 3)	(M2, 4)	(M0, 2)

この表の意味するところは，以下の通りである．すなわち，仕事 J0 は始めマシーン M0 を 3 時間使い，以降 M1 を 3 時間, M2 を 4 時間，最後に M3 を 1 時間使う．J1・J2 に関しても同様である．これら 3 つの仕事を平行して行うとき，処理時間を考えよう．

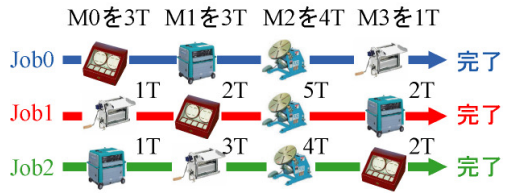

図 23 JSSP の問題 (T は時間単位)

Job0・Job1・Job2 の順に完成させるようにスケジューリングした場合どうなるか？ただし，空いているマシーンがあったら，適宜使うものとする．これを遺伝子 000011112222 と表す．0 が 4 つあるが，1 つ目の 0 は M0 を 3T 使う作業, 2 つ目の 0 は M1 を 3T 使う作業を意味し，以下，同様である．

作業と時間の関係は下記のガントチャートで見ることができる．この場合，全 Job を完了に 21T かかることが分かる．（図 24の上側）

一方，別の実行手順 001122221100 の場合, 18T しかかからない．（図 24の下側）

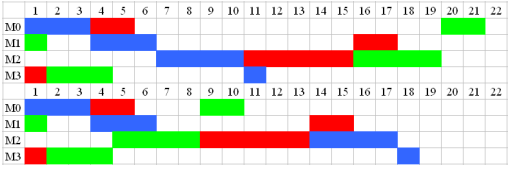

図 24 ガントチャート(000011112222/001122221100)

この問題のように，3Job×4Machine の遺伝子の組み合わせは，3 万 4650 通り程度である．しかし，Job の数を 1 つ増やしただけ(4Job×4Machine)で 6306 万 3000 通りになり，さらに Job の数を倍(すなわち 6Job×4Machine)でおよそ 3246 兆通りとなり，もはや網羅的な試験は不可能であろう．ちなみに 1 ケースを 1 秒で試験したとしても 3264 兆秒で，これはおよそ 1 億年に相当する．

網羅的な探索をして最悪時間のテストケースを求めることは事実上不可能である．したがって，従来であれば，ランダムにテストケースを作成し，その反応時間を測定し，最悪実行時間を守れているかを検証していた．

GA を使えば，実行したテストケースで時間がかかったテストケースを掛け合わせて，より優秀な，つまり時間のかかるテストケースを導出し効果的なテストケース生成できる．

JSSP に対して，本来であれば GA は最良時間すなわち最短時間で全 Job を完了させる組合せを最適解として求めるものであるが，ここでは適応度の考え方を全く逆にしているのである．

4.2. 適用事例

今回，下記のような 7 つの仕事・9 つのマシーンの JSSP の問題を設定した．

表 4 JSSP の問題

	(仕様マシーン，時間数)			
J0	(M5, 5)	(M4, 2)	(M7, 3)	(M1, 1)
J1	(M2, 8)	(M5, 10)	(M8, 4)	(M0, 2)
J2	(M4, 10)	(M2, 8)	(M7, 5)	(M0, 10)
J3	(M5, 2)	(M8, 6)	(M7, 2)	(M1, 8)
J4	(M8, 9)	(M5, 1)	(M7, 1)	(M0, 7)
J5	(M0, 4)	(M5, 8)	(M2, 6)	(M6, 5)
J6	(M1, 1)	(M2, 9)	(M3, 8)	(M8, 6)
(M2, 9)	(M3, 8)	(M6, 3)	(M0, 1)	(M8, 10)
(M4, 1)	(M3, 6)	(M6, 6)	(M7, 5)	(M1, 9)
(M6, 3)	(M1, 5)	(M3, 5)	(M8, 8)	(M5, 4)
(M0, 7)	(M2, 1)	(M4, 9)	(M6, 6)	(M3, 5)
(M6, 8)	(M1, 4)	(M4, 8)	(M2, 4)	(M3, 6)
(M3, 4)	(M4, 7)	(M8, 4)	(M1, 2)	(M7, 7)
(M7, 6)	(M0, 5)	(M4, 2)	(M6, 8)	(M5, 8)

200 個のテストケースを作るにあたって，次の 2 つの

手法を採用した.

A. サイズ 20 の集団を, ランダムに 10 回つくる

B. サイズ 20 の集団をランダムに 1 回つくり,

以降は GA を使って 9 世代の子孫の集団をつくる

ただし, GA については, 次の手法を選んだ

適応度: 実行時間

選択: ルーレット選択

交叉: 2 点交叉(付録参照)

突然変異:なし

各集団の適応度(すなわち実行時間)の平均値を算出すると, 表 5のようになった. GA によって明らかに実行時間のかかるテストケースが多く生成されたことが分かった.

表 5 各集団の実行時間の平均値

	ランダム生成	GA による生成
1 回目	90.0	90.2
2 回目	90.5	91.5
3 回目	92.4	90.4
4 回目	91.6	90.2
5 回目	91.5	91.0
6 回目	91.7	92.3
7 回目	92.3	94.4
8 回目	92.7	94.6
9 回目	91.3	96.7
10 回目	90.4	99.5

5. 結論と今後の課題

第4.2節の表 5で示した通り, 確かに GA によって世代を経るごとに, 最悪時間に近い(実行時間のかかる)テストケースを多く生成することができた.

ただし, GA 生成側の最終第 10 世代における 20 個の個体について, (完全に同じ遺伝子型を持つものは存在しないものの)類似性の高い個体が多く存在していた. 今回, 3.3.3の突然変異や3.4.1の図 20の棲み分けの手法を採用しなかったこと, 問題設定自体が簡単すぎたことなどの理由が考えられるが, いずれにせよ「集団の多様性の維持」という意味ではランダム生成に勝ることはできない.

それでも, 適応度の低い個体を淘汰するメカニズムは効率的な検証には欠かせないものであり, 表 4よりも遥かに複雑な問題に対してこそ, GA の威力はより発揮できるはずである.

効果的なパラメータの設定(3.2.1参照)や動的計画法(2.3参照)など他の手法との併用によって, GA には常に改善の方向が見出せる.

いつでも必ず厳密解が得られるわけではないからこそ, いつでも改良・拡張・発展を目指すことができることが, メタヒューリスティク(meta-heuristic)たる GA の魅力であり醍醐味なのである.

ソフトウェアのテスト現場において, 効率のよいテストケース生成法をお探しの方は, 是非 GA を試してほしい.

参考文献

[1] 伊庭斉志. 遺伝的アルゴリズムの基礎. オーム社 (1996 年 9 月).

[2] John H. Holland. Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence. Bradford Books (April 1992)

[3] 平野 廣美. 応用事例でわかる遺伝的アルゴリズムプログラミング. パーソナルメディア (1995 年 12 月)

[4] 伊庭 斉志. Excel で学ぶ遺伝的アルゴリズム. オーム社 (2005 年 11 月)

[5] 小嶋和徳／石亀昌明／松尾広. 遺伝的アルゴリズムにおける SRG 選択法の提案. 情報処理学会論文誌 Vol.40 No.12 (1999 年 12 月)

[6] 大場 和宏・田崎 栄一郎. 騙し問題解決のための拡張型GA によるニューラルネットワークの学習. 第 18 回 医療情報学連合大会 18th JCMI(1998 年 11 月)

[7] 森 隆史. スキーマ定理. 知的システムデザイン研究室 第 59 回月例発表会(2003 年 6 月)

[8] 立木秀樹. グレイコードと実数. 共立出版 bit 2000 年 1 月号

[9] Newton (ニュートン) 2008 年 12 月号 ニュートンプレス(2008 年 10 月)

目時伸哉

1994 年 東京大学・理学部・数学科卒業.

2000 年 東京大学大学院・数理科学研究科・後期博士課程修了. 博士(数理科学).

現在, キャッツ組込みソフトウェア研究所研究員.

付録. JSSP の遺伝子型の交叉方法

図 7の巡回セールスマン問題 TSP や第4節で取り上げたスケジューリング問題 JSSP などを GA で扱う場合，遺伝子は単なるビット列ではなく，より複雑な構造と制限がある．TSP の場合，遺伝子は重複のない自然数の順列である．また 3Job×4Machine の JSSP の場合，遺伝子は 0 か 1 か 2 を 4 個ずつ全 12 個並べた順列である．したがって，3.3.2で述べた交叉をそのまま適用しても，不適切な遺伝子（致死遺伝子）ができてしまう．このような場合，適切な交叉法を採用する必要がある．ここでは，3Job×4Machine の JSSP の遺伝子の交叉法を説明する．

1. まず交叉する 2 点をランダムに選ぶ
2. 選んだ 2 点で挟まれた部分を左側から見ていき，共通する遺伝子があったら 1 組ずつマークを付ける
3. マークの付いた遺伝子を，位置も込めて入れ替える
4. 残った遺伝子は順序を保って戻す

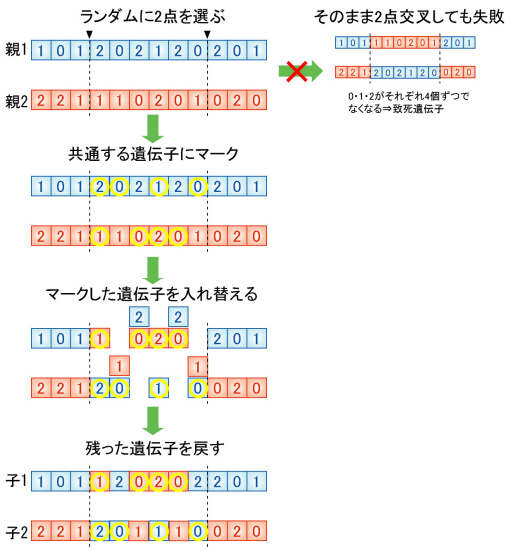

図 25 JSSP の遺伝子交叉法