遺伝的アルゴリズム (genetic algorithm)

目次

1 遺伝的アルゴリズム

1.1 概要

遺伝的アルゴリズム (genetic algorithm) は 1960 年代に Holland によって導入されたアルゴリズムである。これは生物の進化の過程を模したアルゴリズムであり、1980 年代ごろから活発に研究され始めた。特徴としては主に以下の 3 つがある。

- 複数の解候補を同時並行的に更新していく反復解法
- 解候補に対する交叉 (crossover) という演算が用いられる
- 柔軟性が高い

遺伝的アルゴリズムの登場以来、このアルゴリズムの特徴を取り入れて別のアルゴリズムが改良されたり、遺伝的アルゴリズムが別のアルゴリムを取り入れて改良されたりしている。

遺伝的アルゴリズムは生物の進化過程である遺伝を模したアルゴリムである。生物の個体は形質とよばれる形態的・生理的な特徴を継承していく。これは細胞が分裂するさいに細胞中の遺伝子 (gene) が複製されることによるものである。細胞分裂時には各遺伝子が一列に並んだ染色体 (chromosome) となり、各遺伝子が染色体上のどの位置に並ぶかは決まっており、この位置を遺伝子座 (locus) という。また遺伝子は劣勢な遺伝子や優勢な遺伝子等複数種類存在し、遺伝子座が同じであり種類の違う遺伝子を対立遺伝子 (allele) と呼ぶ。対立遺伝子の並びを遺伝子型 (genotype) と言い、遺伝子型と遺伝子の種類によって表面的に表われる形質を表現型 (phenotype) という。通常の細胞分裂では遺伝子型は変化しないが、2 本の染色体の一部が入れ換って遺伝子の組替えが生じたり、一部の遺伝子が変化したりすることによって新しい遺伝子型が形成される。ここで前者を交叉、後者を突然変異 (mutation) と呼ぶ。このような手順で生物は子孫を残す。生物いろいろな遺伝子を持つが、環境に適応し生き残ることができた個体が子孫を残すことになるため、子孫に遺伝する遺伝子は環境への適応度 (fitness) が高いものである可能性が高い。このように優秀な遺伝子が残される仕組みを選択・淘汰 (selection) と呼ばれる。

遺伝的アルゴリズムは上述した生物の進化過程の上で個体を解候補、適応度を目的関数へと対応づけ、選択・交叉・突然変異に対応した演算を実行することで目的関数により適した解を得ようとするアルゴリズムである。このアルゴリズムは生物の進化をモデル化したものではあるが、最適化の計算過程と生物進化の知見とは強い関係はない。

1.2 アルゴリズム

一般に遺伝的アルゴリズムは最適化問題に対する近似解法として用いられる。目的関数を z、実行可能領域を A としたときに A を終域とする全単射が定まるような始域 P の元として個体が定まる。個体 $p \in P$ は遺伝子 g の集まりとして表現され、どのように遺伝子を並べて個体を表現するかは比較的任意性がある。また 1 列に並んだ遺伝子 (?) を染色体と呼ぶ。最も単純な並べ方としては目的関数の変数を 1 次元的に並べたものであるが、交叉・突然変異等が適切に働くのであれば別の表現を用いることもできる。

また遺伝子による個体の表現 $p \in P$ を遺伝子型、遺伝子型から得られる解候補 $a \in A$ を表現型と言い、P の元から A の元へと写像することをデコーディング (decoding) という。これらの関係を図式に表すと

$$P \xrightarrow{decoding} A$$

となる。本節では個体と遺伝子型を等しいものとして扱う。

複数の個体 (individual) からなる個体群 (population)P に対して、以下の繰り返し計算を行うことで解を探索する。なお、遺伝的アルゴリズムでは 1 回の繰り返しを 1 世代 (generation)

と呼ぶ。

Step1　初期個体群を生成し、世代を 1 とする

Step2　交叉則・突然変異則・選択則を現在の個体群に適用し、新たな個体群を生成する

Step3　終了条件を満たせば計算を終了しそれまでに得られた最もよい個体を解とする、そうでない場合は世代を 1 つ増やして Step2 へ戻る

Step3 での終了条件は一般に世代数の上限として定められる。ほかには、何世代か続けて現最良個体よりもよい個体が生成されないこととする等の方法がある。

Step2 での交叉則・突然変異則・選択則は遺伝演算子 (genetic operation) と呼ばれる。

交叉則　複数の個体の遺伝子をいくつか入れ換えて新しい個体を生成する演算である。ある $p_1, p_2, \cdots, p_n$ に交叉則を作用させ、$p'_1, p'_2, \cdots, p'_n$ が得られたとき、$p_1, p_2, \cdots, p_n$ それぞれを親、$p'_1, p'_2, \cdots, p'_n$ それぞれを子という。

突然変異則　ある個体の遺伝子を変更して新しい個体を生成する演算である。交叉則と同様に、p から p' が生成された場合は p を親、p' を子という。

選択則　個体群中の個体、もしくは個体群中の個体から交叉則や突然変異則によって生成された子から次の世代に必要な個体を選択する演算である。各個体への目的関数が大きい (小さい) 個体、もしくは適応度の高い個体を選択する。

交叉則や突然変異則によって生成された子は親と近い遺伝子を持つため、親の適応度が高ければ子の適応度も高くなることが期待される。また交叉則において親の適応度が高くない場合でも親の優れた遺伝子のみを受け継ぎより高い適応度の子が生成される可能性もある。

各個体が 6 つの遺伝子で構成される 6 個体の個体群に対する遺伝的アルゴリズムの一例を示す。

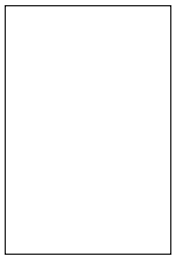

図 1　遺伝的アルゴリズムの動作例

図の説明

図 1 での例のみでは実際にアルゴリズムを実行することはできない。これは具体的な交

叉や選択則を示していないからであるが、これらは適当に定める必要がある。この任意性によって遺伝的アルゴリズムは様々な問題へと適用できる柔軟性を持っている。また定め方次第で得られる解や収束速度に大きく影響する。基本的には最適化問題の特徴をうまく取り出して、それをアルゴリズムに組込むことで良い結果が得られやすい。以降、比較的よく用いられる手続きを説明する。

1.2.1 個体表現

遺伝的アルゴリズムの設計において最も重要なのが個体の表現方法である。個体の表現が重要なのは他のアルゴリズム等でも同じであり、問題の特徴をうまく取り入れる必要がある。したがって、他の問題で用いられる表現方法を用いることも可能である。また、問題の特徴を取り入れる必要があるため個々の問題に依存する。よって、いくつかの最適化問題の例と共に説明する。

0-1 表現　遺伝子を 0 もしくは 1 とし、個体を遺伝子が 1 次元的に並んだもとして表現する方法であり、最もよく利用される。

ナップザック問題 (knapsack problem)　ナップザック問題は最大化問題であり、N 個の荷物をどのようにナップザックへと入れるかを決める問題である。具体的には、各荷物 i $(i = 1, 2, \cdots, N_1)$ に価値 T_i とコスト C_i が与えられており、コストが一定値を越えないように最大の価値となるように荷物を選ぶ問題である。コストの最大を C とするとナップザック問題は

$$\begin{aligned} \max z &= \sum_{i=1}^{N} x_i T_i \\ \text{s.t.} \sum_{i=1}^{N} x_i C_i &\leq C \\ x_i &\in \{0, 1\} \end{aligned}$$

と書かれる。

このナップザック問題では 0 と 1、すなわち x_i を i が 1 から N なで並べることで個体を表現する。

関数最適化問題　関数最適化問題は関数の最小値を求める問題である。ここでは

$$\begin{aligned} \min z &= \sum_{i=1}^{N} x_i^2 \\ \text{s.t. } -5.12 &\leq x_i < 5.12 \end{aligned}$$

と書かれる最小化問題の個体表現を考える。

本来 x_i は実数値であるが、0-1 表現とするために x_i を 4 ビットの 2 進数で表わすこととすると個体は 0 もしくは 1 を $4N$ 個並べたものとなる。ある x_i を表わす各ビットを $b_i^1, b_i^2, b_i^3, b_i^4$ とすると

$$x_i = -\frac{5.12 + 5.12}{16} \sum_{j=0}^{3} b_i^j 2^j + 5.12$$

という変換によって x_i を得る。

遺伝子の列として b_i $(i = 1, 2, \cdots, 4N)$ が与えられたとき、各 x_j $(j = 1, 2, \cdots, N)$ へのデコーディングは

$$x_j = -\frac{5.12 + 5.12}{16} \sum_{k=1}^{4} b_{4(j-1)+k} 2^{k-1} + 5.12$$

として行なわれる。

0-1 表現の拡張　0-1 表現では 0 もしくは 1 を遺伝子として用いたが、そもそも 0 と 1 である必要はないので代替として A や B を用いたりすることもできる。また、0 と 1 の 2 つである必要もないので 3 種類以上の遺伝子を用いる場合もある。

0-1 のナップザック問題ではナップザックが 1 つであったがナップザックが複数の場合もあり、ナップザックの数を M とし j 個目のナップザックの容量を C_j とすると

$$\begin{aligned} \max z &= \sum_{i=1}^{N} x_i^j T_i \\ \text{s.t. } \forall j,\ &\sum_{i=1}^{N} x_i^j C_i \leq Cj \\ \forall i,\ &\sum_{j=1}^{M} x_i^j = 1 \\ x_i^j &\in \{0, 1\} \end{aligned}$$

と書くことができる。このような場合遺伝子として $\sum_{j=1}^{M} x_i^j 2^{j-1}$ を用いることで遺伝子の種類は $1, 2, \cdots, M$ の M 種類となる。

最適化問題では 2 進数で表わすのでなくそもそも遺伝子を $-3.84, 0.64, 0$ として個体表現を行うこともできる。

$f(x)$ あ $f(x)$ い

順序表現　最適化問題には最短経路探索等解が順序として与えられるものが存在する。このような問題での個体表現には順序そのものを用いることが多い。0-1 表現と違い、各対立遺伝子は個体のうちに 1 つづつしか存在することができない。

巡回セールスマン問題 (traveling salesman problem)　ある枝にコストが与えられたグラフに対してコストを最小化するオイラー閉路を得る問題である。巡回セールスマン問題は全てのノードを通る路を解とする問題であるため、解となる路の長さはノード数と等しくなる。

フローショップ・スケジューリング問題 (flow shop scheduling problem)　フローショップ・スケジューリング問題は N 個の製品と M 個の機械があり、どの製品も機械 1 から機械 M へと順に入れることで生産される。しかし各機械は同時に 1 つの製品しか生産することができず、また同時に 1 つしか機械を動かすことができない。この条件下で機械 1 にどのような順番で製品を入れれば生産の合計時間を最小化できるかを考えるものである。この場合も巡回セールスマン問題と同じように製品の並びによって個体を表現することができる。

その他の表現　0-1 表現、順序表現共に 1 列の遺伝子、すなわち 1 つの染色体による個体表現を例としたが、個体は複数の染色体で表現することもできる。また、列でなくグラフや行列を用いることもできる。さらに列である場合も遺伝子の数が固定である必要もない。特殊な染色体に対しては、その染色体の表現用の交叉則や突然変異則を用いたりする。

1.2.2　交叉則

交叉則は遺伝的アルゴリズムの特徴的な演算である。この交叉則が有効に機能することでほかの解法よりもよりよい解が得られる場合がある。交叉則において重要なのでは個体表現が持つ元問題の特徴を残して子へと継承することである。そうでない場合はランダムに子を生成しているのと変わらない。交叉は世代の全ての個体に対して行われるのでなく、事前に定めた確率によって実行される。また、交叉する時にも乱数が用いられる。

以降、一般に用いられる 2 個の親から 2 個の子を生成する交叉の例を示す。

1 点交叉 (one-point crossover)　1 点交叉は 0-1 表現や拡張された 0-1 表現に対して用いられる交叉である。この交叉を順序表現に対して用いると順序表現が持つべき同じ遺伝子が存在してはならないという上限条件が崩れてしまうため、順序表現には用いることはできない。実際の生物では、2 つの染色体のある部分以降が交換されることで新たに染色体が作られる。この実際の生物をモデルとしたものが 1 点交叉である。

1 点交叉ではまず、染色体のうちのある 1 箇所をランダムに選びそこから染色体を 2 つにわけ部分遺伝子を生成する。わけた部分遺伝子を入れ換えることで新たな子の染色体を 2 つ生成する。

2 点交叉 (two-point crossover)　2 点交叉は 1 点交叉を拡張したものである。1 点交叉では 1

箇所をランダムに選びそこで染色体を 2 分したのに対し、2 点交叉ではランダムに染色体上の 2 箇所を選び、選んだ 2 箇所に挟まれている染色体を入れ換える。

一様交叉 (uniform crossover)　一様交叉は 1 点交叉は 2 点交叉等をより拡張したものである。一様交叉では各遺伝子座にてランダムに遺伝子を入れ換える。これは毎回交叉点の数がランダムに変化するような n 点交叉ということができる。

部分一致交叉 (partially matched crossover)　部分一致交叉は順序表現に対して用いられる交叉である。具体的には以下の手順で行われる。

Step1　親において、ランダムに 2 つの遺伝子座 $m_1, m_2 (m_1 < m_2)$ を選び、それらの遺伝子座の後ろに切れ目を入れる。

Step2　親 1 の 2 つの切れ目の外側の部分遺伝子列を子 1 にコピーし、親 2 の 2 つの切れ目の間の部分遺伝子列を子 1 にコピーする。また、$n = m_1 + 1$ とする。

Step3　子 1 の左から n 番目にある対立遺伝子が子 1 の中に 1 つしかない場合は、n 番目の遺伝子を親 1 の n 番目の遺伝子に置き換える。対立遺伝子が 2 つある場合は子 1 の中の n 番目でない同じ対立遺伝子の遺伝子座を探し、その位置の遺伝子を親 1 の n 番目の遺伝子で置き換える。

Step4　もし $n < m_2$ ならば $n = n + 1$ として Step3 へ戻る。

Step5　親 1 を親 2、親 2 を親 1、子 1 を子 2 に読み替えて Step2 から Step4 までの手順を繰り返し実行して子 2 を得る。

例として遺伝子を 3 分割する方法を示したが、一様交叉や n 点交叉のようにして分割して行うこともある。

順序交叉 (order crossover)　順序交叉は部分一致交叉と同様に順序表現に対して用いられる交叉である。遺伝子の長さが L である個体群に対しては具体的には以下の手順で行われる。

Step1　個体の遺伝子座を 2 つランダムに選び、$m_1, m_2 (m_1 < m_2)$ とする

Step2　親 1 の遺伝子 I_1 を遺伝子座 1 から m_1 までの遺伝子を I_1^l、$m_1 + 1$ から m_2 までの遺伝子を I_1^c、$m_2 + 1$ から L までの遺伝子を I_1^r とし、同様に親 2 の遺伝子 I_2 を I_2^l, I_2^c, I_2^r とわける。

Step3　I_1^c 内での対立遺伝子同士の順序関係が親 2 の遺伝子 I_2 内での順序関係と一致するように I_1^c 内の遺伝子を並び変えて $I_1^{c\prime}$ とする。

Step4　同様に I_2^c 内での対立遺伝子同士の順序関係が親 1 の遺伝子 I_1 内での順序関係と一致するように I_2^c 内の遺伝子を並び変えて $I_2^{c\prime}$ とする。

Step5　I_1^l と $I_1^{c\prime}$ と I_1^r を並べることで子 1 の遺伝子とし、I_2^l と $I_2^{c\prime}$ と I_2^r を並べることで子 2 の遺伝子とする。

部分一致交叉と同様に、一様交叉や n 点交叉のようにして分割して行うこともある。

1.2.3 突然変異則

突然変異則も交叉則と同様に遺伝的アルゴリズムの特徴的な点であり、問題の特徴を破壊せずに子へと継承するような演算であるべきである。また、突然変異則はタブーサーチ等ほかの解法でも同じ手法が用いられることもあるため、別の解法から流用することも可能である。

0-1 表現等に対する突然変異則　ランダムに一つ遺伝子を選択し、その遺伝子を別の種類の対立遺伝子へと変更する。変更する遺伝子の数は複数の場合もある。

交換突然変異 (swap mutation)　交換突然変異は順序表現等で個体の順序表現を崩さないように突然変異を起こす。この突然変異はランダムに選択した 2 つの遺伝子を交換するものである。

移動突然変異 (shift mutation)　移動突然変異も交換突然変異と同様に順序表現等に対して用いる突然変異である。この突然変異はランダムに選択した 1 つの遺伝子を、ランダムに選択したもう 1 つの遺伝子の前に移動させるものである。

1.2.4 選択則

選択則は優秀な個体を選択するように設計する。一般に目的関数の値が良いものを選べばよいが、目的関数の値が悪い値であっても良い遺伝子を持っている可能性はあるため、そのような個体を選べたほうがよい。しかし、どの遺伝子が優秀かは一般に事前にわからないため乱数を用いる。

一般に適応度は正の数であり、大きければ大きいほどよいものであるとする。最大化問題では目的関数をそのまま使えばよく、最小化問題では目的関数に負数を乗じ、一定の正数を足すことで適応度として用いることができる。また、最大化問題であっても一定の正数を加算する場合がある。

適応度比例選択 (fitness proportional selection)　実際の生物の適応度はその個体の繁栄度合いとみることができる。よって生物のモデルをそのまま遺伝的アルゴリズムに適用したのが適応度比例選択 (もしくはルーレット選択) である。適応度比例選択では個体群 S のある個体 $s \in S$ が選択される確率 e_s は、s の適応度を F_s とすると

$$e_s = \frac{F_s}{\sum_{s' \in S} F'_s}$$

と書かれる。ここで適応度 F_s $(s \in S)$ にある定数 $B > 0$ が足されていてある F'_s を用

いて

$$F_s = F'_s + B$$

と書かれるとすると選択される確率は

$$e_s = \frac{F_s}{\sum_{s' \in S} F'_s} = \frac{F'_s + B}{\sum_{s' \in S} F'_{s'} + NB}$$

と書くことができ、$B \to \infty$ で極限をとると

$$\lim_{B \to \infty} e_s = \lim_{B \to \infty} \frac{F'_s + B}{\sum_{s' \in S} F'_{s'} + NB} = \frac{1}{N}$$

となる。また、e_s は B に対して単調減少関数であるため B が大きくなるにつれて F_s に関わらず e_s の値は一定へと近づくことがわかる。つまり B が大きければ大きいほど適応度が低い個体も選ばれやすくなるということになる。この B に関して、B の値が小さいときに選択の圧力が強いといい、B の値が大きくなると選択の圧力が弱くなるという。

選択の圧力は遺伝的アルゴリズムの性能に大きく影響するため、適応度比例選択を用いる場合は選択の圧力が適切になるように定める必要がある。

トーナメント選択 (tournament selection)　適応度比例選択では、選択の圧力を適切に定める必要があり、また実行時によい個体が選択されない可能性もある。このデメリットを解消する代表的な方法がトーナメント選択である。

トーナメント選択では個体群 S からランダムに $T \in \mathbb{N}$ 個の個体を取り出し、そのなかから値が最もよい個体を選択することで個体を 1 つ選ぶ。この操作を $|S|$ 回行うことで $|S|$ 個の個体を選ぶ。これにより、適応度の高い個体だけでなく適応度が低い個体が選択される可能性も高くなる。このトーナメントのサイズ T は一般に 2 がよく用いられる。

またこの方法だとトーナメントに参加できない個体が発生する可能性があるため、個体群 S の個体を T 個ならべて $|S|T$ 個の個体の集りを作り、これを並び変えて T 個の集りを再度作ることでサイズ T のトーナメントを $|S|$ 個作る方法も用いられる。この方法であれば全ての個体が必ずトーナメントに参加することができるため適応度が高い個体は必ず選択される。

適応度比例選択では適応度から計算される確率に従って個体が選択されるのに対して、トーナメント選択では適応度の大小とトーナメントのサイズに従って個体が選択される。

1.2.5 その他の設計

遺伝的アルゴリズムを行う際に決めなければならないのは交叉則や突然変異則、選択則だけではない。通常は個体数 S、世代数の上限 G、交叉則における交叉率 P_c、突然変異率 P_m を定めなければならず、選択則によっては選択の圧力やトーナメントのサイズを定める必要もある。また、初期個体の生成を行う際に問題の特徴から優れた個体が生成できるのであれば必要な世代数を少くすることもできる。

一般に個体数 S や総世代数 G は大ければ最適解が得られやすいが、計算時間も必要になる。個体数 S は数十から数百くらいの値がよく用いられ、総世代数 G は問題やアルゴリズムの構成によるが求められる計算時間に合わせて設定される。交叉則は遺伝的アルゴリズムの主となる演算であるため、交叉率 P_c は 0.6 以上の大きめの値が用いられる。一方、突然変異率が高いとランダムに個体を生成するのと変わらないため、突然変異則 P_m は小さめの値として 0.1 以下とされる場合が多い。

また、よりよい解を得るために交叉則・突然変異則・選択則だけでなく別の操作を行う場合がある。それらの操作の例を説明する。

エリート保存 (elite preserving)　これまで示した選択則では個体を選ぶだけであったため、交叉則や突然変異則によって前の世代の個体群での個体と一致する個体は一般に次の世代に表われない。これでは最良の個体が消えてしまう可能性があるうえに、次の世代で悪い個体しか生成されなかった場合に効率が悪くなる。その対策として選択則と合わせてエリート保存という操作を行う場合がある。

エリート保存では交叉則、突然変異則を適用する前に個体群 S から適応度の高い E 個の個体を除外し個体群 S' を作る。そして得られた個体群 S に対して交叉則、突然変異則を適用して S' と同じサイズの個体群を生成する。次の世代の個体群は S に交叉則、突然変異則を適用して得られた集合に除外した E 個の個体を加えることで生成する。最初に優秀な個体を除外しておくことで、交叉則や突然変異則によって悪い個体しか生成されなくても次の世代では優秀な個体が残っていることになるため、最終的によい解を得ることができると期待できる。

Steady-state 型　適応度比例選択やトーナメント選択を用いる場合は、一般に全ての個体に交叉則や突然変異則を適用する機会を与えることで、全ての個体が入れ替わる可能性を作る。しかし、交叉則や突然変異則を適用しうる個体数に制限をもうけることで 1 世代で入れ替わる個体数に上限を与える方法もある。このような方法で行われる遺伝的アルゴリズムを Steady-state な遺伝的アルゴリズムという。

1.2.6　数値計算例

1.2.7　アルゴリズムの改良

hoge()　hoge

hoge()　hoge

参考文献

[1] 相吉英太郎他，安田恵一郎，“メタヒューリスティクスと応用”，電気学会.