遺伝的アルゴリズムでナップサック問題を解く

鉦谷　龍平　水口　研二
山田　大貴

１．研究概要

ナップサック問題を遺伝的アルゴリズムという手法を用いて解決した。

２．研究の具体的内容

（１）ナップサック問題とは

ナップサック問題とは「入る容量が決まったナップサックと価値と容量を持った品がありナップサックにどれだけ容量が超えないように品を詰め、なおかつ価値の合計が一番高くする組み合わせを探す。」という問題である。

品を 40 個としたとき総当たりで計算すると 2^40 通りとなりおよそ 10 兆通りである。簡単な計算で解くことはできるが 10 兆回計算することになるので総当たり法ではなく効率的に解くために遺伝的アルゴリズムを用いた。

図１ナップサック

（２）遺伝的アルゴリズムとは

遺伝的アルゴリズムとは
人工知能の手法の一つで交叉、突然変異などの処理を繰り返しことで近似値解を求める手法である。

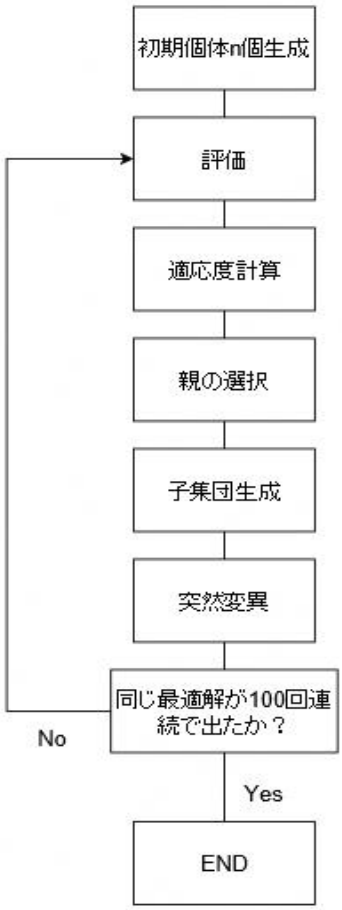


図２　フローチャート

まず、個体を 2 進数の乱数によって 20 個生成する。(0101, 1111 など)個体(乱数の集合体)には遺伝情報がある。個体の持つ遺伝情報 0, 1 は、それぞれ 1 は買う, 0 は買わないを示しており、桁数は品の数を表している。
左から順番に商品を割り当てているため、商品の情報（商品名, 容量, 価格）を与えることによって個体の容量と価値を出すことができる。

例えば　図３のような時飴とバナナに１が立っているため次のようになる。

容量　　20+130=150

価格　　30+120=150

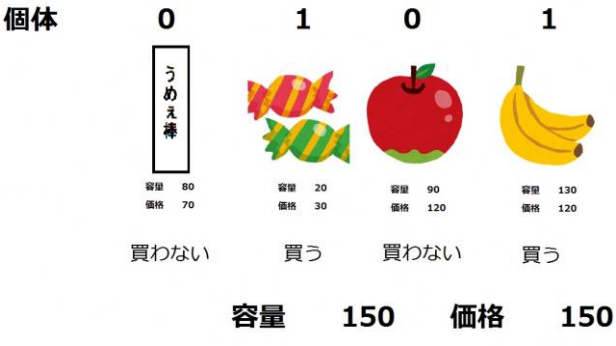
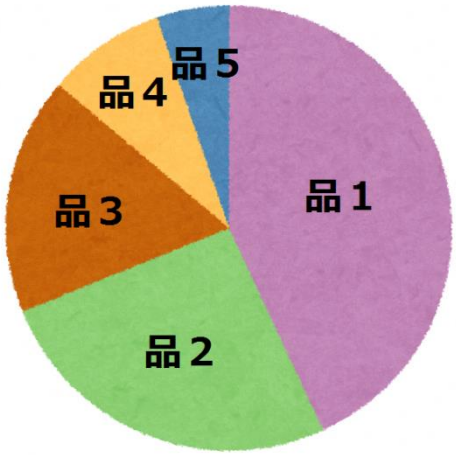


図３　遺伝情報

集団の価値の合計における個体の価値の百分率を適応度とし、適応度が高いほど優秀な個体とする。適応度をもとにルーレット選択という方法で親を生成し、交叉をすることで子集団を生成する。交叉には種類があり、交叉の方法で解を出すまでの時間が変わっていく。今回、私たちが利用した交叉方法は２点交叉である。

（３）ルーレット選択とは

ルーレット選択とは、乱数で１～１００まで数字を出し、それに対応した適応度の個体を選択する方法。



（４）２点交叉とは

まず、交叉とは植物や動物の遺伝情報を入れ替え子供に受けつくことである。遺伝的アルゴリズムでは交叉をすることにより、最適解に近づいていく。二点交叉は親２個体の遺伝情報を２か所で切断し、切断した間の遺伝情報を入れ替え子供を作る交叉の方法で、切断する２か所は乱数によって決める。この方法では一回の交叉に２人しか子供が生まれないため 20 人子供を作るためには 10 回交叉することになる。

下図に二点交叉の例を示す。親１は遺伝子を全て０にし何も買わせず、親２は遺伝子を全て１にし全て買わせる。この２個体を二点交叉で交叉させると図のような子が２個体生成される。

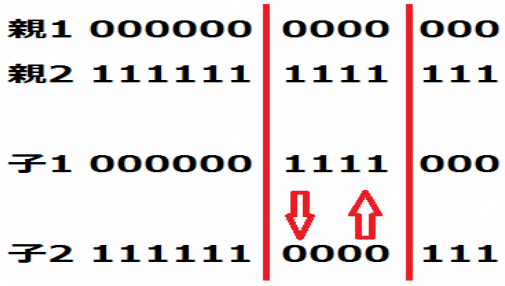


図４　二点交叉

（５）突然変異とは

突然変異とは個体の遺伝情報を商品ごとに３％の確率で反転する。突然変異をすることによって、悪い方向に収束することを防ぎ、最適解により近づける。

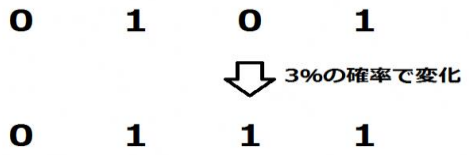


図５　突然変異

突然変異まで処理が終わると遺伝情報が確定し、元の情報は似ているが違う情報になってくる。一回の突然変異と交叉でどれだけ最適解に近づけるかによって終了するまでの時間が変わってくる。次頁に交叉、突然変異後の２個体の例を示す。

0 0 1 1 0 0 1 容量150 価格150

0 0 0 0 0 0 1 容量30 価格30

⇩ 交叉,変異後

0 0 0 1 1 0 1 容量160 価格200

0 0 1 1 0 0 1 容量150 価格150

図 6 遺伝情報の変化

(6)終了条件

終了条件はプログラムを終了するための条件で、条件を厳しくすればするほど最適解に近づいていく。今回は「最適解が 100 世代にわたって更新されなかったときに終了」としている。

右図の実行結果は各世代で結果を出力させ値を表示している。図 9 の 1000 世代の結果と図 10 の 10000 世代の結果が同じことから、世代が重なるにつれて、合計金額や最適解が同じ値になり最適解が収束しているのがわかる。

(7)プログラム言語

3 人はそれぞれ違うプログラミング言語を使いナップサック問題を解いた。言語の種類は VC#, VC++, VB を使い、VC#は鉦谷、VC++は水口、VB は山田でプログラムを作成している。違う言語を使っているが交叉の方法、突然変異の方法などは統一しており、違う言語で作っている人でも見ればわかるようにしてあるため、互いにアドバイスができる用になっている。

３．研究のまとめ

遺伝的アルゴリズムを使いナップサック問題の最適解を求めさせた。遺伝的アルゴリズムは初期集団を乱数で作るのでは当てずっぽうのような答えになる(図 7)。世代が増えるにつれて最適解が大きくなりより正確な答えになっていく。1000 世代になるとほとんど最適解が出るようになった。結果的に人が総当たりで計算すると 19013243 年かかる問題を約数秒で解くことができるようになった。

実行結果

図 7 第 1 世代

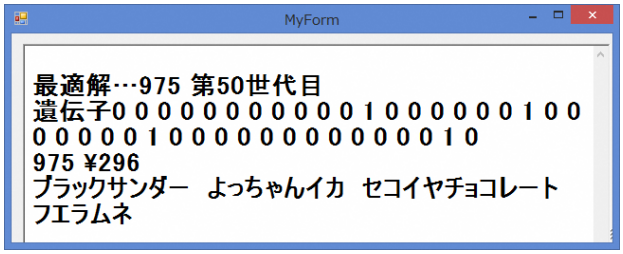


図 8 第 50 世代

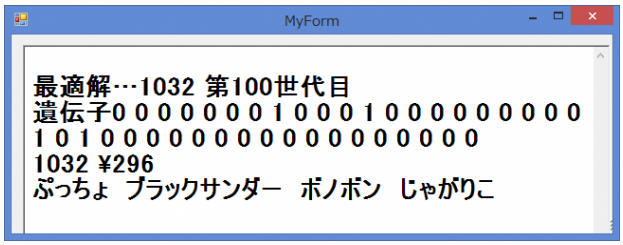


図 9 第 100 世代

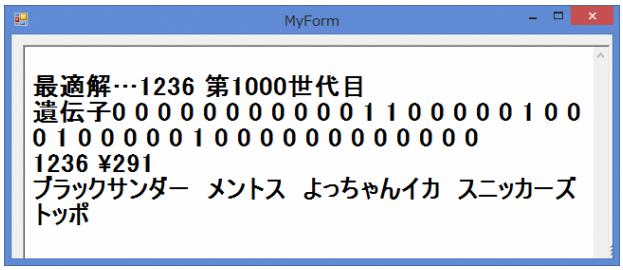


図 10 第 1000 世代

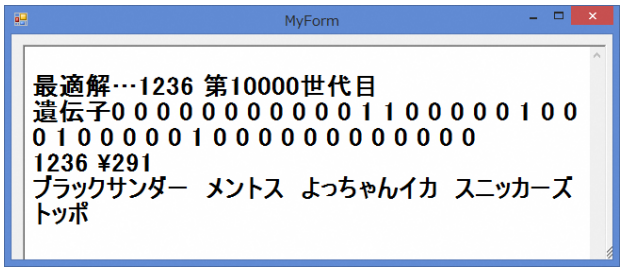


図 11 第 10000 世代

感想

私は今回の研究を始めるまで、遺伝的アルゴリズムとナップサック問題について全く知らなかったので新しいことに挑戦するというのは怖かったがプログラムをくむにつれて新しい知識が増えて良かった。また、C#というC言語の派生を使うことになりFOR文やIF文を書くときはほとんど同じ書き方だが変数や配列の宣言の仕方であったりExcelを簡単に扱えたりなど違うことも多くあり最初は戸惑ったが段々と出来るようになり確認せずとも作れるようになった。だが、変数やプログラムの組み方が汚かったために苦戦することがあったので改善していきたいと思う。

鉦谷龍平　使用言語:C#

遺伝的アルゴリズムというものは予備知識がある程度あったが、それを自分自身の手で活用し使える形にすることは極めて難しかった。

C++の資料や記事も昔のものばかりでまるで古文書を解読しているかのような気分だった。しかし、その技術をだんだんと使いこなせるようになり、自分の力で探し出したものをより確かな知識・技術へと昇華させていくことはとても良い経験であり、おそらく今後も活かされていくだろうと思う。

この課題研究を通して知識が全くない状態から１つのモノを作り出すことの難しさ・魅力を感じることができた。

水口研二　使用言語:C++

去年の先輩方の課題研究を見て安易に遺伝的アルゴリズムを使いたいと思い、始めた研究であったが、実際に研究を始めてみると見た目以上に難しく、遺伝的アルゴリズムを理解するのに相当な時間を要した。

実習で学習していたVBであったが､自分のしたいことが参照しているテキストに書いていないことが多かったので自分でメソッド等を調べるのが大変であった。また、VBはC言語と違い、簡単に書ける分あまり難しいことが出来ないので、簡単なコードで複雑なことをするためにコードを書いているととても読みにくく、メンテナンスのし難いソースコードになってしまった。

この課題研究を通して如何に可読性の高いソースコードを書くことが難しいかということがよく分かった。また、新しいことが多かったので、調べることの楽しさや、新しい知識を自分のものにする嬉しさを感じることが出来た。

山田大貴　使用言語:VisualBasic

参考文献

・Microsoft
https://docs.microsoft.com/ja-jp/visualstudio/#pivot=languages