技術レポート

Java 遺伝的アルゴリズム・パッケージ「jagatara」 －人員配置問題への応用－

ビジネス開発本部　技術研究部　伊藤　強

1．はじめに

現在、技術研究部では、ソフトコンピューティング手法を利用したアプリケーションの可能性について調査を行っている。この中で筆者はソフトコンピューティング手法の１つである遺伝的アルゴリズム（Genetic Algorithm）に注目し、応用実験を行っている。遺伝的アルゴリズムを用いた計算は様々な場面に適用可能であるが、現状ではあまり応用されているとは言い難い。これは遺伝的アルゴリズムを応用する場合に、簡単に使えるツールがほとんど存在しないことが一因と考えられる。そこで筆者は遺伝的アルゴリズム・ツールキット「**jagatara**」を作成した。

このレポートでは、遺伝的アルゴリズムの概略、**jagatara** についての説明、実際的な問題に応用した例として人員配置問題への適用実験の報告を行う。

2．遺伝的アルゴリズム

2.1　遺伝的アルゴリズムとは

遺伝的アルゴリズム（以下「GA」と略す）は1970年代半ばに、ミシガン大学の John Holland とその研究室によって提案された手法である［Holland96］。GA は自然界における生物進化のメカニズムを模倣した手法である。自然界では複数の個体からなる集団が親から子へと世代を重ねていくことで環境に適応するように進化していく。GA はこの過程を模倣し、複数の解候補を遺伝的な操作を加えることで進化させ、求める解に近づけていく計算方法である。

生物学では遺伝子情報を遺伝子型と呼び、その発現形態を表現型と呼ぶ。同様に GA でもビット列、文字列として情報を持つ遺伝子型と実際の問題における解候補に対応する表現型を扱う。

GA では下記の手順によって解を求める。

(1)　**初期集団の生成**

複数の個体をランダムに生成し初期集団を形成する。

(2)　**選択**

おのおのの個体の表現型に対して評価関数を適用し、その個体の適応度を求め、交配を行う個体、淘汰される個体を選択する。選択方法には複数の異なる特性をもった提案がある。

(3)　**遺伝操作**

選択された２つの個体から新しい個体を作る交叉、ある確率で染色体の一部を他の値に書き換える突然変異などの遺伝操作を行う。

(4)　**世代交代**

集団に対して選択と遺伝操作を施して新しい集団を形成する。それぞれの集団を世代と呼ぶ。世代を重ねることで優秀な遺伝子を選定していく（図１）。

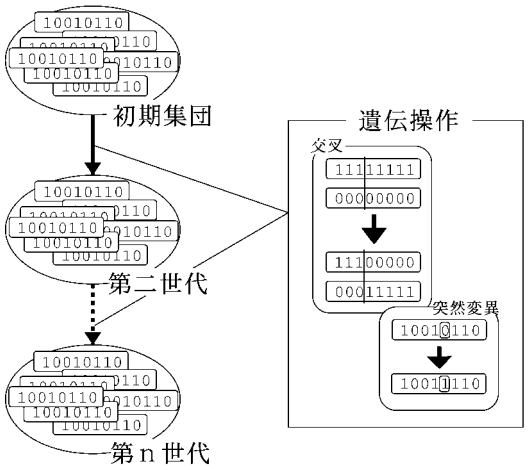

図１　GA

2.2 遺伝的アルゴリズムの特徴

以上GAの理論的な面を説明したが、GAを機能的な面で見ると一般的に次のような特徴がある。

長所として、実用時間内で比較的良い解が得られる点があげられる。組合せが膨大となり実用時間内では計算不可能な問題に対してもGAを利用することで比較的良い解を導き出すことができる。また適用範囲の広さもGAの長所の1つである。GAは解の状態を遺伝子型で表わすことさえできれば、どのような問題でも適用可能である。

短所としては、パラメータの設定や遺伝子型へのマッピング方法に一般的な規範が無いという点があげられる。GAのパラメータには集団の大きさや交叉を行う確率を表す交叉率、突然変異が起こる確率を表す突然変異率など複数あるが、これらの設定は試行錯誤で行わなければならない。

2.3 遺伝的アルゴリズムの適用領域

前項で挙げたように、GAは遺伝子型へのマッピング次第で様々な問題に適用可能である。特に最短経路探索のような探索問題、人員配置問題やスケジュール問題といった組合せ問題に適用されることが多い。これらは比較的素直に問題を遺伝子型にマッピングすることができる点、組合せの数が膨大となって通常の計算では解を導き出すことができない（NP－困難な）問題となる点で共通している。

3．Java GA パッケージ「**jagatara**」

3.1 jagatara 概要

jagatara はJavaで実装したGAのツールキット・パッケージである。**jagatara** はなるべく制約なく、様々な問題に対して利用できるように設計されている。**jagatara** を利用するための作業は下記の4つである。

・Step 1 ：遺伝子型へのマッピング
　GAで計算させたい問題（表現型）を遺伝子型にマッピングする。
・Step 2 ：クラス定義
　jagatara パッケージの中の抽象クラス GA から導出したクラスを定義する。
・Step 3 ：パラメータ設定
　GA クラスでフィールド定義されているパラメータを設定する。
・Step 4 ：評価関数定義
　GA クラスで抽象メソッドとして宣言されている評価関数メソッドを実装する。

以上の作業で準備は全て整う。後は自分のシステム内でGAクラスの導出クラスから、初期集団を生成し、世代交替を繰り返すことで計算を行う。

3.2 jagatara の使用例

ナップザック問題を例に **jagatara** の使用方法を説明する。ここで言うナップザック問題とは「入れることのできる重量に制限がある容器（ナップザック）と、重さと価値の判っている複数の財があるとき、重さの合計が容器の制限を超えずに価値の合計を最大にする組合せを求める」問題である。ここでは重さの制限が40である容器と、表1の14個の財があるとする。

表1　各財の重量と価値

	財1	財2	財3	財4	財5	財6	財7
重量	8	2	18	11	24	12	15
価値	886	1915	2335	492	421	1027	2059
	財8	財9	財10	財11	財12	財13	財14
重量	13	15	22	11	17	7	12
価値	2926	426	736	368	429	530	1123

(1) Step 1 ：遺伝子型へのマッピング

それぞれの遺伝子が各財に対応して1または0の値を取り、対応する財を容器に入れる／入れないを表わすと決める。したがって染色体の長さは14、遺伝子の取り得る値は2種となる（図2参照）。

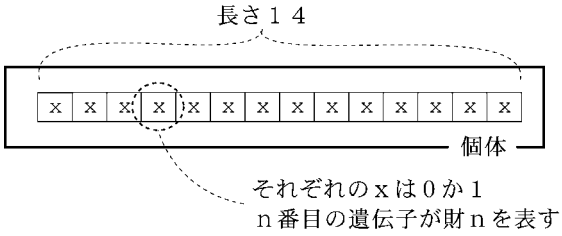

図2　遺伝子コーディング

これによって、{財2と財5と財10が容器に入っている}という表現型は図3の遺伝子型にマッピングされる。

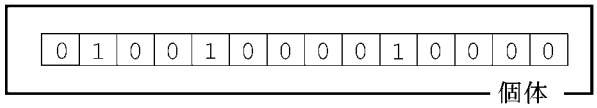

図3　例：遺伝子型

(2) Step 2 ：クラス定義

jagatara パッケージの中の GA クラスから導出したクラス“Knapsack”を定義する（図4）。

```
import jagatara.* ;

class Knapsack extends GA {
    .
    .
}
```

図 4　クラス定義

(3) Step 3 ：パラメータ設定

遺伝子コーディングのパラメータと遺伝操作のパラメータという二種類のパラメータを設定する（図 5 ）。遺伝操作のためのパラメータとは集団の個体の数、交叉を行う確率を表す交叉率、突然変異の起きる確率を表す突然変異率などである。例ではパラメータ設定はコンストラクタ内で行っている。

```
/*Knapsack クラスのコンストラクタ*/
Knapsack(){
    /*遺伝子コーディングのパラメータ*/
    geneCount=2 ;                  //遺伝子の種類
    chromosomeLength=14 ;          //染色体の長さ

    /*遺伝操作のパラメータ*/
    mutationRate=0.05 ;            //突然変異率
    crossOverRate=0.75 ;           //交叉率
    populationSize=100 ;           //集団内の個体の数
}
```

図 5　パラメータ設定

(4) Step 4 ：評価関数の定義

評価関数は GA クラスにおいて抽象メソッドとして宣言されているため、これを実装する（図 6 ）。これは個体クラスのインスタンスを引数に取り、倍精度浮動少数点数の適応度を返すメソッドである。

個体中の遺伝子には（個体インスタンス）. chromosome（＜染色体の番号＞）. gene（＜遺伝子の番号＞）. value（）というメソッドによりアクセスすることができる。これを用いて遺伝子の値が 1 である財の重量（配列 WEIGHT に格納されている）と価値（配列 VALUE に格納されている）の合計を計算し、重量の合計が制限（定数 LIMITWEIGHT）を超えていれば 0 、そうでなければ価値の合計を適応度として返している。

```
public double evaluateFunction(Individual indiv){
  int  weight=0,value=0 ;

  for(int i=0 ; i < chromosomeLength ; i++){
     if(indiv.chromosome(0).gene(i).value()==1){
        weight += WEIGHT[i] ;
        value += VALUE[i] ;
     }
  }

  if(weight > LIMITWEIGHT)value=0 ;

  return value ;
}
```

図 6　評価関数

(5) Step 5 ：システムへの組み込み

ここまでのステップを踏むことにより準備は整う。後は GA を導入するシステムに組み込みを行うだけである。ここでは“Knapsack”クラスのメインメソッドで実際の計算を記述する（図 7 ）。

```
public static void main(String[] args){
   //Step2で定義したクラスのインスタンスを作成
   Knapsack ga = new Knapsack() ;

   //作成したインスタンスを引数に集団クラスのインスタンスを作成
   Population p = new Population(ga)；

   for(int i=0 ; i < GENERATIONCOUNT ; i++)
      p = p.reproduction() ; //世代交代を行う

   for(int i=0 ; i < ga.chromosomeLength ; i++)
      System.out.print(
         p.elite(0).chromosome(0).gene(i).value()) ;

   System.out.println() ;
}
```

図 7　メインメソッド

例では定数 GENERATIONCOUNT で定義した回数の世代交代を繰り返して、最終世代の最優良個体を表示する（最良個体はelite(0)というメソッドで取ることが可能）。

(6) Step 6 ：実行

図 8 にコンパイルから実行、実行結果の表示までを示す（“$”はコマンドプロンプトを表している）。解として遺伝子型で“11000011000000”つまり「財 1 ，財 2 ，財 7 ，財 8 の組合せ」を出力している。

このとき重量の合計は38、価値の合計は7786となった。これは全探索で検証した結果、各財の組合せのうち重量制限40の中で価値が最大となるものであった。

```
$javac Knapsack.java
$java Knapsack
11000011000000
$
```

図 8　実行

3.3 jagatara の特徴

3.1項、3.2項では主に使用方法について **jagatara** の説明を行ったが、ここでは特徴について詳しく述べる。

⑴ **柔軟な遺伝子コーディング**

パラメータを変更することで、１つの遺伝子の表わす値の種類や、１本の染色体中の遺伝子の数、１つの個体中の染色体の数などを簡単に変更できる。

⑵ **選択、交叉、突然変異、世代交代**

選択方法は基本的に、それぞれの適応度に比例した確率で個体が選択されるルーレット戦略を用いている。また、パラメータの変更により指定した数の優良個体を残すこともできる。

交叉方法は一点交叉、任意数点での多点交叉、一様交叉をパラメータの変更により選ぶことができる。交叉率はパラメータにより設定する。

突然変異率もパラメータで設定する。各遺伝子ごとにこの値で突然変異が適用され、遺伝子が書き換えられる。ただし、突然変異で書き換える遺伝子の値は適用前の値に関わらずランダムに決定する。すなわち、突然変異が適用された遺伝子の値が適用前と必ず異なるわけではない。

⑶ **表現型からの初期集団の生成**

通常、初期集団の個体のもつ遺伝子はランダムに生成される。しかし、巡回セールスマン問題など、この方法では上手くいかない場合も存在する。このような場合でも表現型の個体をランダムに生成し遺伝子型に変換することで初期集団を生成できることが多い。**jagatara** ではこのような表現型からの初期集団の生成も抽象クラス `PhenoType` の実装によって行うことができる。

⑷ **可変長染色体 GA**

可変長染色体 GA は単純 GA の持つ、非線形問題に対する脆弱性を補うために提案された手法である［Goldberg89］。単純 GA では染色体中の各遺伝子が表わす情報はその位置により決まる。当然すべての個体は同じ染色体長を持つ。これに対し、可変長染色体 GA では染色体の遺伝子が表わす情報は遺伝子の位置によって決まらない。このため染色体長も一定にならず、遺伝子の表わす情報は対となるシンボルによって決定される。**jagatara** ではこの可変長染色体をパラメータの設定により利用できる。

⑸ **島型移民モデル GA**

単純 GA は世代を重ねて最適解を探索するという性質上、膨大な反復計算を必要とし、その計算コストはかなり高くなる。このため GA を並列・分散型で行うことで処理速度を向上させる手法が多く研究されてきた。この中で単純 GA における集団を複数の副集団に分割し、それぞれの副集団内で独立した遺伝操作を行い、定期的に副集団間で個体の交換を行う手法が提案されている。副集団間での個体の交換は移民と呼ばれ、この手法は島型移民モデル GA と呼ばれている。

構造としての島は、環境が地理的に分離されている典型例である。島型移民モデルは島のように他の環境から分離される複数の副集団を構成し、個々の副集団を単位として独立した進化計算を行わせる。この環境の異なる副集団による独立な進化と移民により、計算速度の向上だけでなく、GA の精度向上において重要な要素である多様性を維持することができる。このため１台の計算機上においても島型移民モデルは有効に働く。**jagatara** でも集団クラスのインスタンスを複数持つことで１台の計算機上で島型移民モデルを利用できる。

4．応用例－スキル要件による人員配置

GA および **jagatara** の実用性検証のため、実問題に対する **jagatara** の応用実験を行った。この章ではこの実験とその結果について述べる。

4.1 スキル要件による人員配置問題

次の基本データを用いてプロジェクトの人員配置を行う問題を考える（図 9 参照）。

・候補者データ－（個人，人員コスト）の組合せ
・保有スキルデータ－（個人, スキル, レベル）の組合せ

人員配置においては下記の要件を満たすものとする。

・メンバーはそれぞれ必要とされるスキルのレベルを満たす
・メンバーのコスト合計を最小にする
・すべてのメンバーは超過アサインされない

例えば、あるプロジェクトで業務設計をするメンバーが２人、DB 管理者が１人、Java のプログラマが５人必要な場合を考える。

このような場合、例えば Java プログラマの中ではリーダーとなるレベルの高いメンバーが１人とそれをサポートする中程度のレベルのメンバーが１人、残りの３人はレベルに拘らないという構成が考えられる。このような組合せをプロジェクトごとの要件データとして使用する。

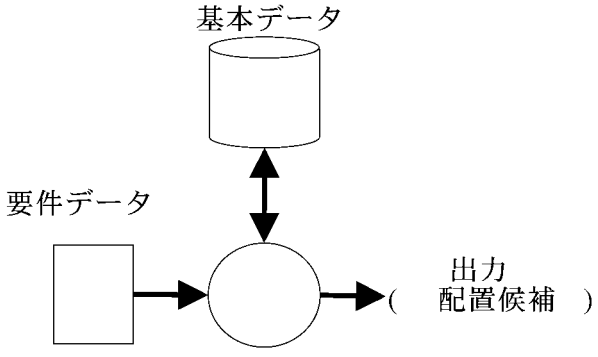

図 9　システム構成

4.2 GAの適用

この問題を計算するため下記のように遺伝子型へのマッピングを行った。

・遺伝子はある候補者が何割アサインされるかを表わし、0～10の値を取る。
・染色体の長さは候補者の数であり、染色体中の遺伝子の位置はそれぞれの候補者を表わす。
・１つの個体はプロジェクトに必要なスキルの数の染色体を持つ。

評価関数では下記の方法で個体の適応度を求めた。

・レベルを完全に満たす組合せならば減点なし、前後のレベルで満たす組合せならば減点１、前後２レベルで満たす組合せなら減点２……。
・人員コストが大きい程、適応度は小さい。
・１人の候補者を10割より多くアサインしていれば減点。

パラメータは下記のように設定した。また、今回の適用では精度の向上を狙い、島型移民モデルを採用した。

・世代交代は3000回。
・各250の個体を持つ５つの副集団。
・交叉率、突然変率などは副集団ごとに異なる。
・移民のトポロジーは循環型、移民間隔は200世代、移民先はランダムに決定。

4.3 結果と考察

今回の実験では要件データとして､表２のデータを用いた。

表２　要件データ

	中レベル	低レベル
販売実績管理	２人	
IBM-PC サーバー		１人
Win２K サーバー	１人	
WebLogic	１人	
Java	２人	５人

プロジェクトのメンバーと成り得る候補者は約1000人を対象にしているが、プロジェクトで必要なスキルを１つも保有していない者はGA計算に入る前に排除される。このため、実際にGA計算に利用された候補者は300人程度であった。

(1) 計算結果について

実験では同じ基本データ、要件データを用いて100回の試行を行った。結果として最適解（求める組合せ）を出力した回数は50回である。求めたメンバーの内１人（Java）の人員コストが最適解より１単位だけ大きかった結果が35回であった。残りの15回に関しても実用上問題にならない範囲の結果を得ることができた。

また、１回の試行にかかった時間は約20分であり、延べ3.75×10^{6}個体を計算をしている。単純にすべての組合せ（300から14を選ぶ組合せ）を計算した場合、約4×10^{24}回数の計算をする必要がある。１回の計算がGAでの一個体の計算と同じだけ時間がかかると仮定すると、全ての組み合わせを計算するには4×10^{16}年かかることになる。

このように実用的な解を短時間で求めることができたことで、GAの有用性を示せた。

(2) 精度の向上についての考察

最適解を出力した試行での、最初に最適解を獲得した世代に注目する。図10は最初に最適解を獲得した試行を、獲得した世代順に並べたたグラフである。縦軸は獲得世代を表し、横軸は獲得した順番である。

このグラフを見ると、最適解を獲得した試行のうち、80%の試行が世代交代の後半である1500世代以降に最適解を獲得している。さらに20%の試行では終了間近の2500世代以降に最適解を獲得している。このことから世代交代を3500回や4000回に増やすことで、より一層の精度向上が予想できる。

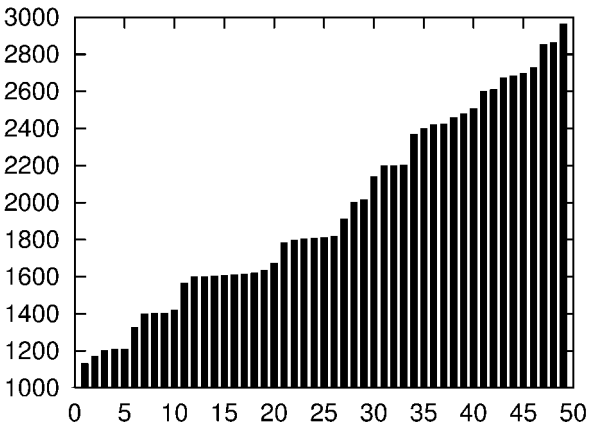

図10　最適解獲得世代

(3) 島型移民モデルについての考察

また、図10のグラフは階段状になっているが、この段差は200世代ごとに行われる移民の後にできている。この事実から、島型移民モデルが効果的に働いていることがわかる。

この移民の効果は集団の平均適応度からも見ることができる。

図11のグラフは最も早く最適解を出力した試行の平均適応度である。グラフのｐ１～ｐ５はそれぞれ副集団を表しているが、移民の後に急激に平均値を上げる副集団がある。例えば副集団ｐ５は１回目の移民である200世代目からの値の変化が顕著である（図12参照）。さらにこの集団は４回目の移民である1000世代目の直後にも急激に平均値を上げ、1004世代目には最適解を獲得する。

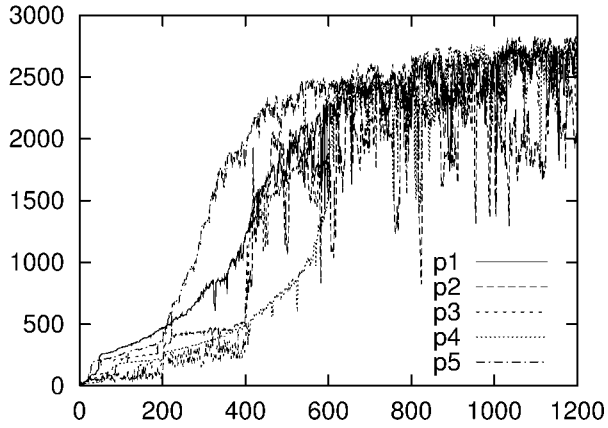

図11　全副集団

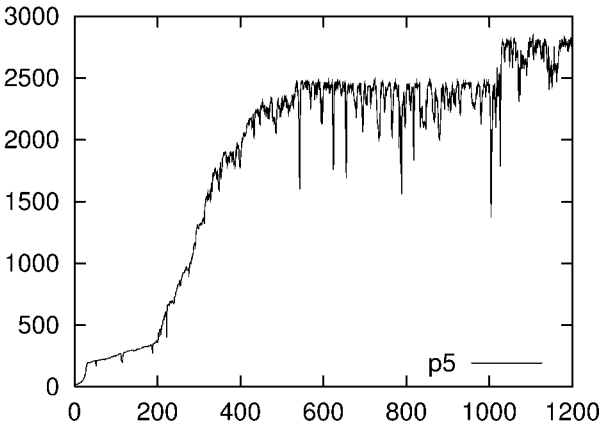

図12　最適解獲得副集団

※図11、図12とも縦軸は評価値の値、横軸は世代数を表わす。

以上から今回の実験に島型移民モデルを採り入れたことは成功だったことがわかる。

5．今後の開発計画

jagatara の機能改良／追加として下記のようなものを予定している。

⑴　複数の VM を利用した分散並列計算

現在の島型移民モデルで行っている疑似並列計算は、多様性の維持という点では有効である。しかし、本来の複数の計算資源を利用して計算時間を短縮するという点では逆効果になってしまう。これは各副集団にある程度の数の個体を割り当てなければならず、全体としての個体の数が大きくなるためである。このため、大きな計算資源を必要とする問題にも対応できるように、複数の JavaVM 上で分散並列計算する機構を追加する。

⑵　パラメータフリー GA の導入

jagatara の特徴は遺伝子型へのマッピングとパラメータ設定、評価関数の実装だけで簡単に GA 計算を行えるという点である。しかし GA の特徴で述べたようにパラメータの設定方法には一般的な規範はなく試行錯誤をともなう作業である。このパラメータの設定作業無しに GA 計算を行う拡張手法がパラメータフリー GA である。この機構を導入し、より扱いやすいツールを目指す。

⑶　その他拡張手法の導入

シミュレーテッド・アニーリング（焼きなまし法）などとのハイブリット手法を中心にその他の拡張手法の導入も検討している。

6．おわりに

Java GA パッケージ「**jagatara**」の使用方法とその応用実験について報告した。技術研究部では、ビジネスシステムのためのインテリジェント・コンポーネントとして、今後も **jagatara** を拡張していく計画である。

〈参考文献〉

1．［Goldberg89］D. Goldberg, B. Korb and K. Deb: Messy Genetic Algorithms: Motivation, Analysis and First Results, TCGA Report 89903（1989）

2．［Holland75］John H. Holland: Adaptation in Natural and Artificial Sytems, University of Michigan Press（1975）

3．［Holland96］John H. Holland: Adaptation in Natural and Artificial Sytems, University of Michigan Press（1996）
邦訳：嘉数侑昇監訳,『遺伝的アルゴリズムの理論』, 森北出版（1999）