

2006年9月1日

SPEA2(ver. 1.0.1)仕様書

同志社大学 工学部 知識工学科
廣安知之
三木光範
金美和
鈴木和徳
吉井健吾

目次

1	はじめに	2
2	プログラムの使用方法	2
2.1	ファイルの構成	2
2.2	コンパイルとプログラムの実行	3
2.3	構築パラメータ	3
2.4	対象問題の設定方法	4
2.5	出力ファイルの見方	6
2.6	入力ファイルを用いた初期化	7
3	クラスの説明	7
3.1	Archive クラス	7
3.2	Crossover クラス	8
3.3	Evaluation クラス	8
3.4	GAInitializer クラス	8
3.5	GrayCoder クラス	8
3.6	Individual クラス	8
3.7	MatingSelection クラス	9
3.8	MTRand クラス	9
3.9	Mutation クラス	9
3.10	OffspringGenerator クラス	10
3.11	OutputUnit クラス	10
3.12	Parameter クラス	10
3.13	Penalty クラス	10
3.14	Popualtion クラス	10
3.15	RandomGenerator クラス	11
3.16	Repair クラス	11
3.17	SPEA2ranking クラス	11
3.18	Truncation クラス	12
4	補足資料	13

1 はじめに

本ドキュメントは、Zitzler の多目的遺伝的アルゴリズム SPEA2 に制約条件を扱うメカニズムを組み込んだプログラムの使用方法と仕様についてまとめたものである。

2 プログラムの使用方法

2.1 ファイルの構成

フォルダ SPEA2 内は以下のような構成になっている。

Makefile

multiobjectiveProblem 多目的最適化のテスト問題が含まれる

- Function.h
- DEB.h
- BPF.h

spea2 SPEA2 のプログラム

- Archive.h
- Archive.cpp
- Crossover.h
- Crossover.cpp
- Evaluation.h
- Evaluation.cpp
- GAInitializer.h
- GAInitializer.cpp
- GrayCoder.h
- GrayCoder.cpp
- Individual.h
- Individual.cpp
- MatingSelection.h
- MatingSelection.cpp
- mtrand.h
- mtrand.cpp
- Mutation.h
- Mutation.cpp
- OffspringGenerator.h
- OffspringGenerator.cpp
- OutputUnit.h
- OutputUnit.cpp

- Parameter.h
- Parameter.cpp
- Penalty.h
- Penalty.cpp
- Population.h
- Population.cpp
- RandomGenerator.h
- RandomGenerator.cpp
- Repair.h
- Repair.cpp
- SPEA2ranking.h
- SPEA2ranking.cpp
- Truncation.h
- Truncation.cpp

`objarchiveData` 各世代での目的関数アーカイブのデータを保存する

2.2 コンパイルとプログラムの実行

フォルダ SPEA2 の下に Makefile があるので実行する .

```
$ make
```

コンパイル後 , 実行ファイル SPEA2 が生成されるので実行する

```
$ ./SPEA2
```

パラメータの設定に関しては SPEA2/spea2/Parameter.cpp を編集する . パラメータについて次節で詳しく述べる .

2.3 構築パラメータ

本プログラムでは以下のパラメータを設定する必要がある .

- `numObjective` 対象とする問題の目的関数の数
- `populationSize` 母集団サイズ (アーカイブサイズと同じ)
- `geneLength` 染色体長
- `numCrossover` 交叉点数
- `maxGeneration` 終了世代数
- `mutationRate` 突然変異率
- `crossoverRate` 交叉率
- `func` 対象問題

finitial 初期個体の生成方法

SPEA2/spea2/Parameter.cpp を編集しこれらのパラメータを指定する．設定後，make を行い実行ファイルを作成し，実行する．対象問題に関しては対象問題クラスを作成する．作成方法については次節に示す．

2.4 対象問題の設定方法

対象問題を設定するには以下の手順に従って対象問題クラスを作成する．以下の制約条件付テスト関数 DEB を例として用いる．

$$\begin{aligned} \text{Minimize } f_1(x) &= x_1 \\ \text{Minimize } f_2(x) &= (1 + x_2)/x_1 \\ \text{Subject to } g_1(x) &= -9x_1 - x_2 + 6 \leq 0 \\ g_2(x) &= -9x_1 + x_2 + 1 \leq 0 \\ &\begin{cases} 0.1 \leq x_1 \leq 1.0 \\ 0 \leq x_2 \leq 5 \end{cases} \end{aligned} \quad (2.1)$$

ファイルの作成

フォルダ multiobjectiveProblem 内に Function クラスの子クラスとして対象問題クラスのファイル (DEB.h) を作る．

```
#include "Function.h"
class DEB:public Function{
    .
    .
    .
};
```

また Function.h に以下の文を追加する．

```
#include "DEB.h"
```

対象問題のパラメータ指定

作成したクラスのコンストラクタで親クラスである Function クラスの以下のパラメータを初期化する．

- max[] (各設計変数の上限値)
- min[] (各設計変数の下限値)
- designNum (設計変数の数)
- objectiveNum (目的関数の数)
- constraintNum (制約条件式の数)

式 (2.1) の例では以下ようになる．

```

#include "Function.h"
class DEB:public Function{
public:
    DEB(int number_design){
        designNum = 2;
        objectiveNum = 2;
        constraintNum = 2;
        max = new double[designNum];
        max[0] = 1.0;
        max[1] = 5.0;
        min = new double[designNum];
        min[0] = 0.1;
        min[1] = 0.0;
    }
};

```

目的関数評価メソッドの作成

親クラス Function 内の抽象メソッド *void evaluate(double* design, double* f)* を記述する。この関数は設計変数配列 *design[]* を受け取り、目的関数配列 *f[]* を更新するものである。

式 (2.1) の例では以下ようになる。

```

void evaluate(double* design, double* f){
    f[0] = design[0];
    f[1] = (1.0+design[1])/design[0];
}

```

制約条件評価メソッドの作成

親クラス Function 内の抽象メソッド *void evaluateConstraint(double* design, double* g)* を記述する。この関数は設計変数配列 *design[]* を受け取り、制約条件の値配列 *g[]* を更新するものである。制約条件のない問題の場合は「対象問題のパラメータ指定」の際に変数 *constraintNum* を 0 にセットし、本メソッドの中身を記述しない。

式 (2.1) の例では以下ようになる。

```

void evaluateConstraint(double* design, double* g){
    g[0] = -9.0*design[0]-design[1]+6.0;
    g[1] = -9.0*design[0]+design[1]+1.0;
}

```

これらのメソッドを加えて完成したものが以下の DEB.h となる。

```

#include "Function.h"
class DEB:public Function{
public:
    DEB(int number_design){
        designNum = 2;
        objectiveNum = 2;
        constraintNum = 2;
        max = new double[designNum];
        max[0] = 1.0;
        max[1] = 5.0;
        min = new double[designNum];
        min[0] = 0.1;
        min[1] = 0.0;
    }
    void evaluate(double* design,double* f){
        f[0] = design[0] ;
        f[1] = (1.0+design[1])/design[0] ;
    }
    void evaluateConstraint(double* design,double* g){
        g[0] = -9.0*design[0]-design[1]+6.0 ;
        g[1] = -9.0*design[0]+design[1]+1.0 ;
    }
};

```

2.5 出力ファイルの見方

目的関数アーカイブのデータは objarchiveData フォルダに、設計変数アーカイブのデータは vararchiveData フォルダにそれぞれ世代ごとに csv として保存される。一列目から順に目的関数 1, 2 ..., 染色体 1, 2 ... のように記録され、一行につき一個体の情報を示している。Fig. 2.1 に目的関数の数が 2, 染色体長が 6, 個体数が 5 である場合の出力例を示す。

個体 1	0. 891738	0. 204803	110001
個体 2	0. 997081	0. 005829	001101
個体 3	0. 817927	0. 333096	000011
個体 4	0. 871927	0. 239744	010111
個体 5	0. 988317	0. 023232	111100
	目的関数 1	目的関数 2	染色体

Fig. 2.1 出力ファイルの例

2.6 入力ファイルを用いた初期化

全ての手法において設定ファイルを用いて初期個体を指定することができる。初期個体設定ファイル名は、「design_variable.ini」である(変更したい場合には、Archive.h ファイルにおける「INITIAL POPULATION FILE」の定義を書き直す必要がある)。また Parameter.cpp で変数 finitial を 1 にする必要がある。

設定ファイル (design_variable.ini) の例を以下に示す。

```
RunCounter x1 x2 x3 x4 x5
1 -7.501370015 -8.850767853 -6.988393869 4.17963018 6.309546484
2 6.654656179 1.392214229 1.444718664 -3.603129042 3.797540378
3 3.550406736 -1.118594598 2.584324095 -3.938642765 -0.30946518
4 -0.538060596 6.945126972 -7.551823214 4.643517926 4.635494333
5 9.292352094 -6.869138526 2.147970777 7.330347315 1.373406009
6 10 10 10 10 10
```

上記の例のように、列が各変数を、行が各評価ケースを表し、1行で1つの個体初期化表現する。また、第一行目および、第一列目には必ず行名札と列名札を書く必要がある。本プログラムでは第一行目および、第一列目は入力値として読み込まない仕組みとなっている。

実際の利用の場面では、個体数とファイルのデータ行数は必ずしも一致しない場合も考えられる。本プログラムでは、その場合には

- 1) 初期化ファイル中のデータの方が多い ファイル中データの先頭n行を使用
- 2) 個体数の方が多い ファイルのデータでn個体目まで初期化した後、n+1 個体以降は、乱数で初期化のように初期化を行う。

#注意: ただし、設定ファイル (design_variable.ini) に書いた設計変数の数と対象とした問題の設計変数の数は必ず統一しておく必要がある。プログラムの方から設定ファイルにおける設計変数の数と実際の対象としている問題の設計変数の数との整合性についてチェックは一切行っていない。

3 クラスの説明

3.1 Archive クラス

目的関数アーカイブを持ち、ランキング、ペナルティ、エリート保存、メイティング選択を行う。主なメンバをに示す。

Table 3.1 Archive クラス

メンバ変数	vector<Individual> objarchive	目的関数アーカイブ
メンバ関数	Population getSearchPopulation() void updateArchive(Population& pop) void exeranking(Population& pop) void penalty(Population& pop) void preserveArchive(Population& pop) void truncation(vector<Individual>& fitness0) void matingSelection(Population& pop)	アーカイブから次世代の探索母集団を生成 アーカイブの更新 探索母集団とアーカイブにランキング 探索母集団とアーカイブにペナルティを与える エリート保存を行う 端切りを行う メイティング選択を行う

3.2 Crossover クラス

交叉を行うクラス。Crossover クラスは抽象クラス (virtual void crossover(..)) であり、実際には子クラスである nPointCrossover クラスが n 点交叉を行う。主なメンバを Table 3.2 に示す。

Table 3.2 nPointCrossover クラス

メンバ変数	double crossoverRate int numCrossover	交叉率 交叉点数
メンバ関数	void crossover(Individual a,Individual b) static Crossover& getInstance()	個体 a と個体 b を交叉する Crossover インスタンスを返す

3.3 Evaluation クラス

評価を行うクラス。主なメンバを Table 3.3 に示す。

Table 3.3 Evaluation クラス

メンバ関数	void evaluateOne(Individual& indiv) void evaluateDesignValOne(Individual& indiv) static Evaluation& getInstance()	設計変数値から目的関数値を得る 染色体から設計変数値を得る Evaluation インスタンスを返す
-------	---	--

3.4 GAInitializer クラス

個体の初期化の際、染色体を作成するクラス。GAInitializer クラスは抽象クラス (virtual void createOne(..)) であり、実際には子クラス RandomInitializer が初期染色体をつくる。RandomInitializer は初期染色体をランダムに決定する。主なメンバを Table ?? に示す。

Table 3.4 RandomInitilizer クラス

メンバ関数	void createOne(int* gene,int length)	染色体長 length の染色体をつくる
-------	--------------------------------------	----------------------

3.5 GrayCoder クラス

グレイコードを扱うクラス。主なメンバを Table 3.5 に示す。

Table 3.5 GrayCoder クラス

メンバ関数	int graydecode(int* grayString,int length) void grayencode(int n,int length,int* gene) static GrayCoder& getInstance()	bit 長 length の染色体を受けてデコードする n を bit 長 length の染色体としてコード化 GrayCoder のインスタンスを返す
-------	--	---

3.6 Individual クラス

個体を表現するクラス。主なメンバを Table 3.6 に示す。

Table 3.6 Individual クラス

メンバ変数	double* objectiveValue double* designValue double* constraint int* gene double fitness int rank bool isFeasible Individual* parent	目的関数の値 設計変数の値 制約条件の値 染色体 適合度 ランク 実行可能領域内なら true 親個体の情報
メンバ関数	void mutate() void evaluation() void crossover(Individual& i) void repair()	突然変異する 設計変数値, 目的関数値を計算し, 実行可能であるか判定 個体 i と交叉する 親個体に向かって引き戻す

3.7 MatingSelection クラス

アーカイブから次世代の探索母集団を選択するメイティング選択を行うクラス。MatingSelection クラスは抽象クラス (virtual void select(..)) であり, 実際には子クラスである TournamentSelection クラスによって, メイティング選択が行われる。TournamentSelection クラスはアーカイブからバイナリトーナメント選択により次の世代の探索母集団を生成する。主なメンバを Table 3.7 に示す。

Table 3.7 TournamentSelection

メンバ関数	void select(Archive& archive, Population& pop)	バイナリトーナメント選択により探索母集団を生成
-------	--	-------------------------

3.8 MTRand クラス

乱数を発生するクラス。主なメンバを Table 3.8 に示す。

Table 3.8 MTRand クラス

メンバ関数	double genrand() int genrand_int(int n) void sgenrand(unsigned long seed)	0.0~1.0の間で double 型乱数を発生する 0 から (n-1) の間で int 型乱数を発生する 乱数の種をセットする
-------	---	---

3.9 Mutation クラス

突然変異をするクラス。Mutation クラスは抽象クラス (virtual void mutateOne(..)) であり, 実際には子クラスの NormalMutation クラスが突然変異を行う。主なメンバを Table 3.9 に示す。

Table 3.9 NormalMutation クラス

メンバ変数	double mutationRate	突然変異率
メンバ関数	void mutateOne(Individual& indiv) static Mutation& getInstance()	indiv を突然変異させる Mutation インスタンスを返す

3.10 OffspringGenerator クラス

交叉を行う際の個体同士のペアを決めるために母集団のソートを行うクラス。OffspringGenerator クラスは抽象クラス (virtual void makePairs(..)) であり, 実際には子クラスとして NormalCrossover クラスがあり, 1 点交叉もしくは多点交叉を担当する。主なメンバを Table 3.10 に示す。

Table 3.10 OffspringGenerator クラス

メンバ関数	void makePairs(vector<Individual>& pop)	個体配列を隣同士が交叉するように並び替える
-------	---	-----------------------

3.11 OutputUnit クラス

探索結果をアーカイブに保存するクラス。主なメンバを Table 3.11 に示す。

Table 3.11 OutputUnit クラス

メンバ関数	void outputObjArchive(vector<Individual>& archive,int generation)	目的関数アーカイブを出力
-------	---	--------------

3.12 Parameter クラス

パラメータを保持するクラス。主なメンバを Table 3.12 に示す。

Table 3.12 Parameter クラス

メンバ変数	int numObjective int numDesign int populationSize int designLength int geneLength int numCrossover int maxGeneration double mutationRate double crossoverRate Function* func int finitial	目的関数の数 設計変数の数 母集団サイズ 1 設計変数のビット長 染色体長 交叉点数 終了世代数 突然変異率 交叉率 対象問題 初期個体の生成方法
メンバ関数	static Parameter& getInstance()	Parameter クラスのインスタンスを返す

3.13 Penalty クラス

ペナルティのセットを行うクラス。Penalty クラスは抽象クラス (virtual void penalty(..)) であり実際は子クラスの NSGAIpenalty によりペナルティを与える。主なメンバを Table 3.13 に示す。

3.14 Population クラス

探索母集団を表現するクラス。交叉, 突然変異, 評価, 引き戻し操作を行う。

Table 3.13 NSGAIpenalty クラス

メンバ関数	void penalty(vector<Individual>& poparc)	個体群にペナルティを与える
-------	--	---------------

主なメンバを Table 3.14 に示す .

Table 3.14 Population クラス

メンバ変数	vector<Individual> population	探索母集団
メンバ関数	void mutate() void crossover() void repair() void evaluate()	突然変異を行う 交叉を行う 引き戻しを行う 探索母集団を評価する

3.15 RandomGenerator クラス

MTRand クラスを使って乱数を発生するクラス . 主なメンバを Table 3.15 に示す .

Table 3.15 RandomGeneration クラス

メンバ関数	double randomDouble() int randomInt(int min,int max) static RandomGenerator& getInstance()	0.0 ~ 1.0 の間で double 型乱数を発生する min から (max-1) の間で int 型乱数を発生する RandomGenerator のインスタンスを返す
-------	--	--

3.16 Repair クラス

交叉, 突然変異により実行可能領域を外れた個体に対して引き戻しを行う . 各個体は親個体の情報を持っており, その親へ向かって引き戻しが行われる . 主なメンバを Table 3.16 に示す .

Table 3.16 Repair クラス

メンバ関数	void repair(Individual& indiv) static Repair& getInstance()	実行可能領域を外れた個体を親へ引き戻す Repair のインスタンスを返す
-------	--	--

3.17 SPEA2ranking クラス

ランキングを行うクラス . SPEA2 独自の適合度割り当て方法を用いてランク (適合度) を求めて各個体にセットする . 主なメンバを Table 3.17 に示す .

Table 3.17 SPEA2ranking クラス

メンバ変数	void spea2ranking(vector<Individual>& population, vector<Individual>& objarchive)	探索母集団, 目的関数アーカイブをあわせてランキング
-------	--	----------------------------

3.18 Truncation クラス

端切りを行うクラス．アーカイブサイズを超えた `vector<Individual>` を渡すと，アーカイブサイズまで個体を削減して `vector<Individual>` を返す．主なメンバを Table 3.18 に示す．

Table 3.18 Truncation クラス

メンバ変数	<code>vector<Distance> obj</code>	全個体同士の目的関数空間距離を保持
メンバ関数	<code>objtruncation(vector<Distance> over)</code>	目的関数空間端切り

4 補足資料

Fig. 4.1 にクラス図を示す .

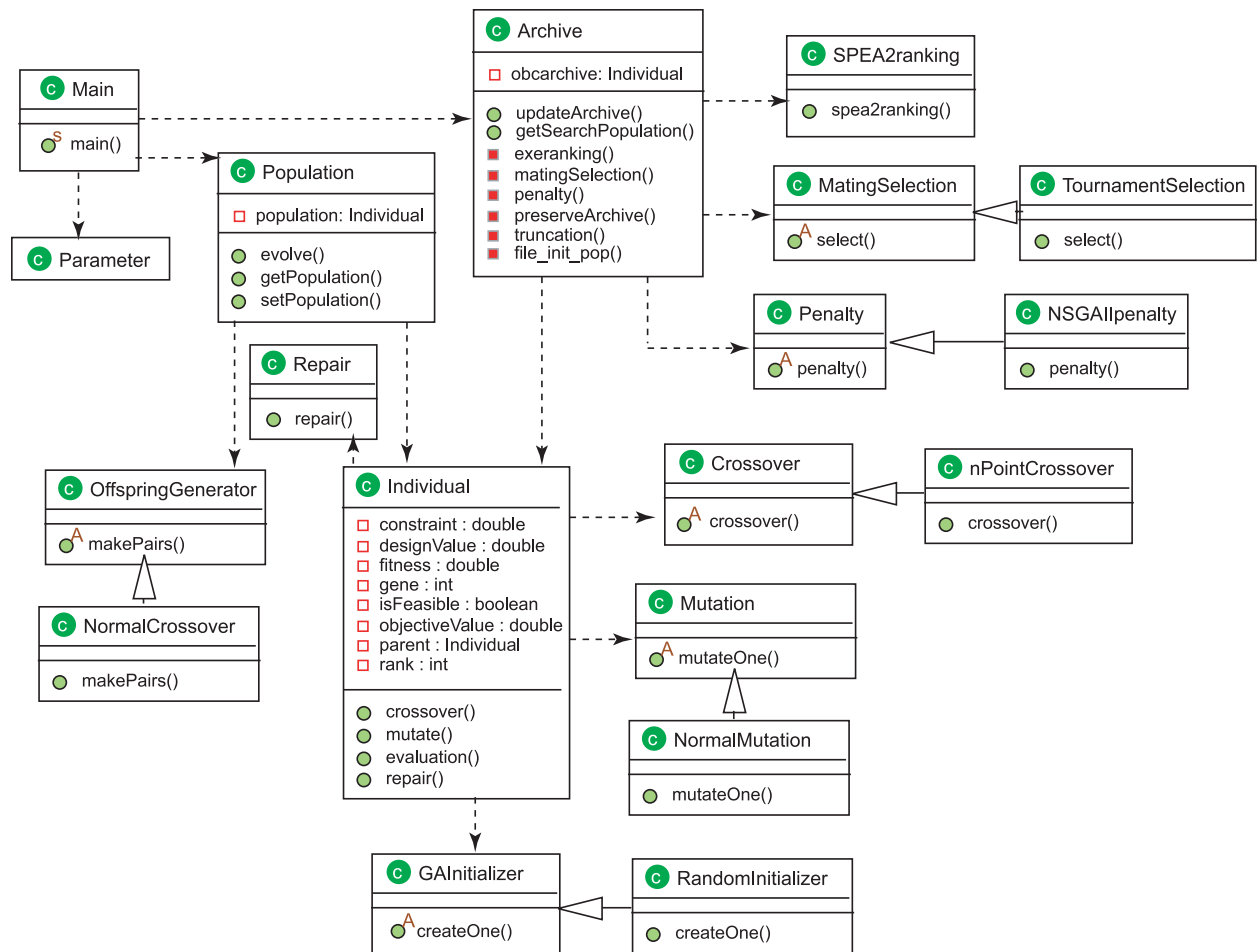


Fig. 4.1 クラス図

Fig. 4.2 にシーケンス図を示す.

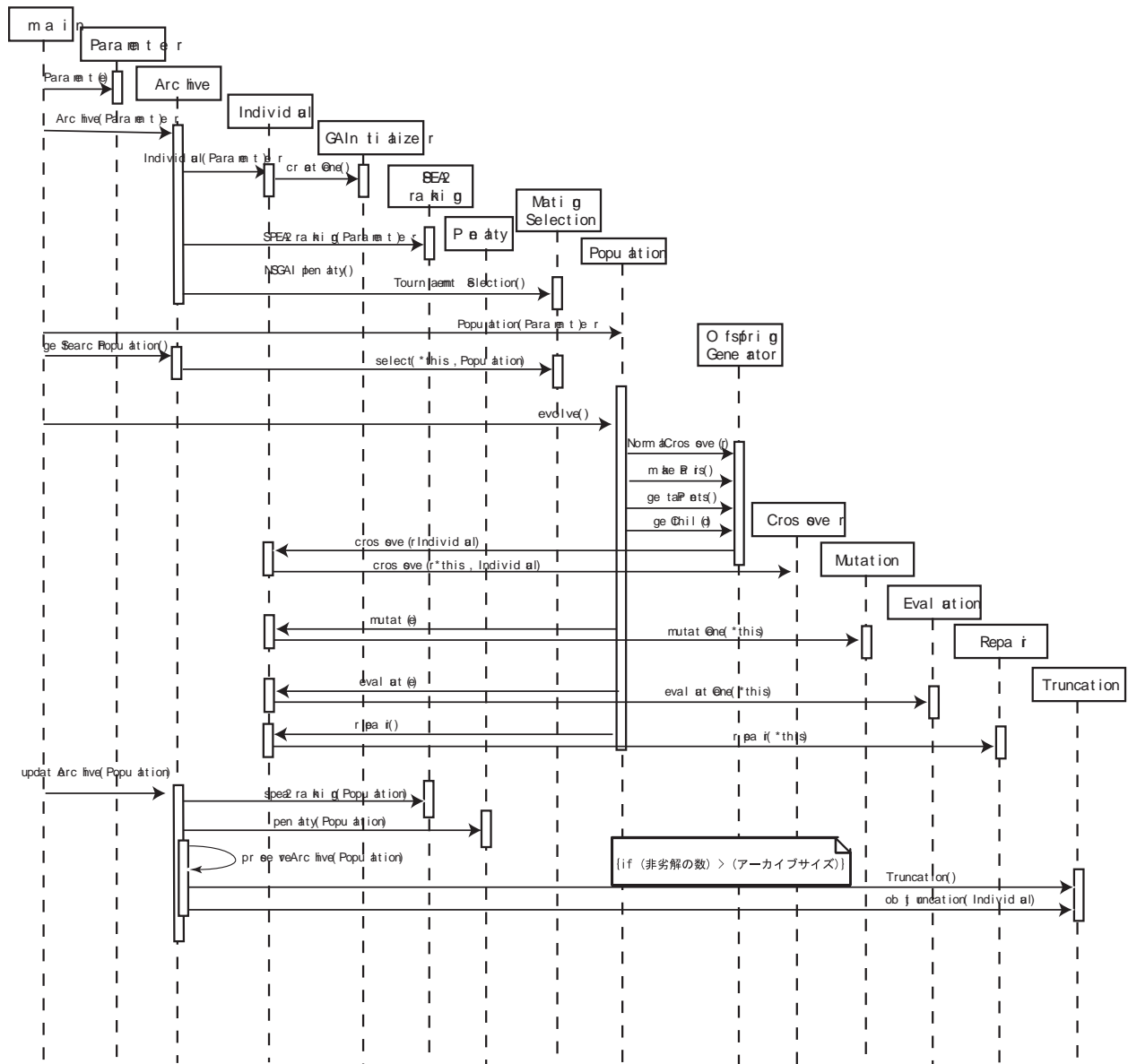


Fig. 4.2 シーケンス図

更新履歴

2006.09.1 ver. 1.0.1

クラス図の一部変更 .

- Main.cpp : Archive クラスから getSearchPopulation() 関数により探索母集団を生成するように変更 .
- Population.cpp : ga_operation() 関数を evolve() に変更 .

- Archive.cpp : getSearchPopulation() 関数を作成. メイティング選択をこの関数内で使用するように変更.
- ControlCrossover.h : クラス ControlCrossover を OffspringGenerator に名称変更.
- OffspringGenerator.h : クラス NPointCrossover を NormalCrossover に名称変更

2006.05.17 ver. 1.0

SPEA2 開発開始 .

Copyright

SPEA2:

Multi Objective Genetic Algorithm Team of Intelligent Systems Design Laboratory, Doshisha University.

Copyright(C) 2006 Tomoyuki Hiroyasu, All rights reserved.

Copyright(C) 2006 Mitsunori Miki, All rights reserved.

Copyright(C) 2006 Mifa Kim, All rights reserved.

Copyright(C) 2006 Kazunori Suzuki, All rights reserved.

Copyright(C) 2006 Kengo Yoshii, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of the authors should not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SPEA2 manual:

Copyright(C) 2006 Tomoyuki Hiroyasu, All rights reserved.
Copyright(C) 2006 Mitsunori Miki, All rights reserved.
Copyright(C) 2006 Mifa Kim, All rights reserved.
Copyright(C) 2006 Kazunori Suzuki, All rights reserved.
Copyright(C) 2006 Kengo Yoshii, All rights reserved.