情報工学実験3：進化計算

(week3) 拡張GA紹介，コードから読むGA

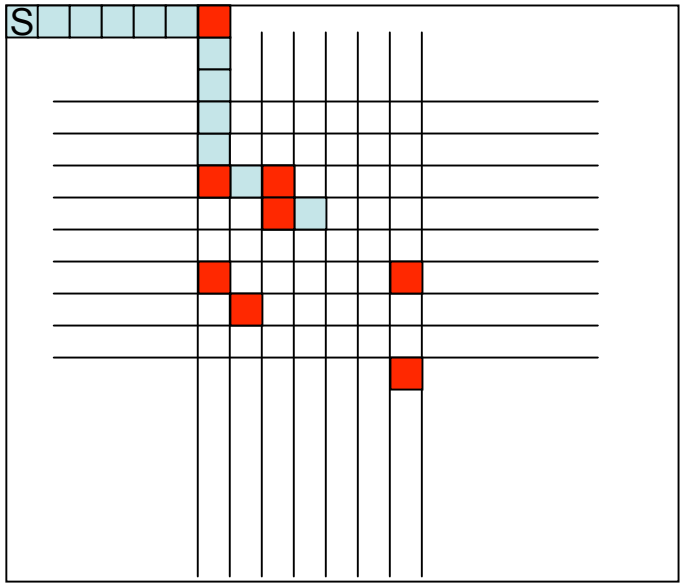
1. GA拡張
    1. 遺伝的プログラミング（GP）
    2. インタラクティブGA
2. コードから読むGA
3. システム開発に向けた準備
4. 今後の進め方

http://www.eva.ie.u-ryukyu.ac.jp/~tnal/2005/info3/
http://www.eva.ie.u-ryukyu.ac.jp/~tnal/Job/GA/Readme.html

遺伝的プログラミング(GP)

- 発想:プログラムそのものを進化(自動生成)させたい.
  - プログラムをコーディング
  - LISPの**S式→木構造**
  - 遺伝子型サイズは**可変長**
- 基本的なフローチャートは同じ.

- 人工蟻による餌探索問題

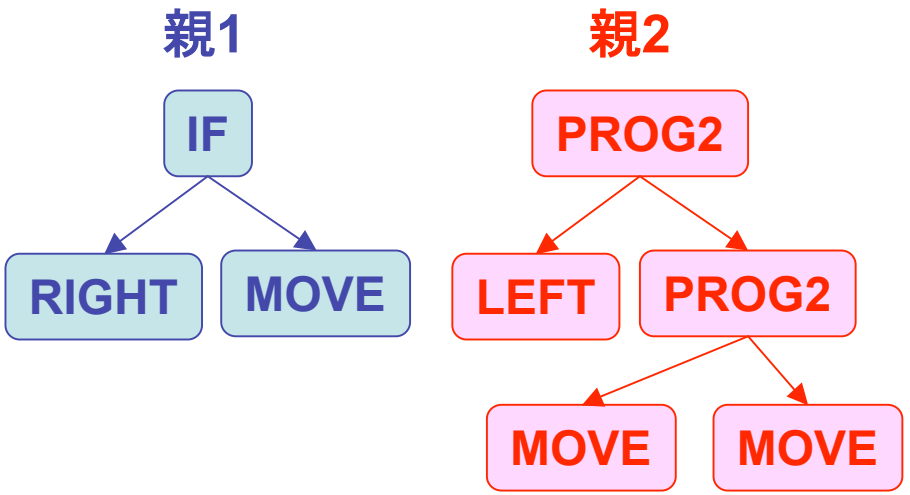


•どのルートが最短?
•餌の配置が変わったときも最適に動ける?

Q: どうやって性能評価すべきか?

餌探索問題(Santa Fe Trail)

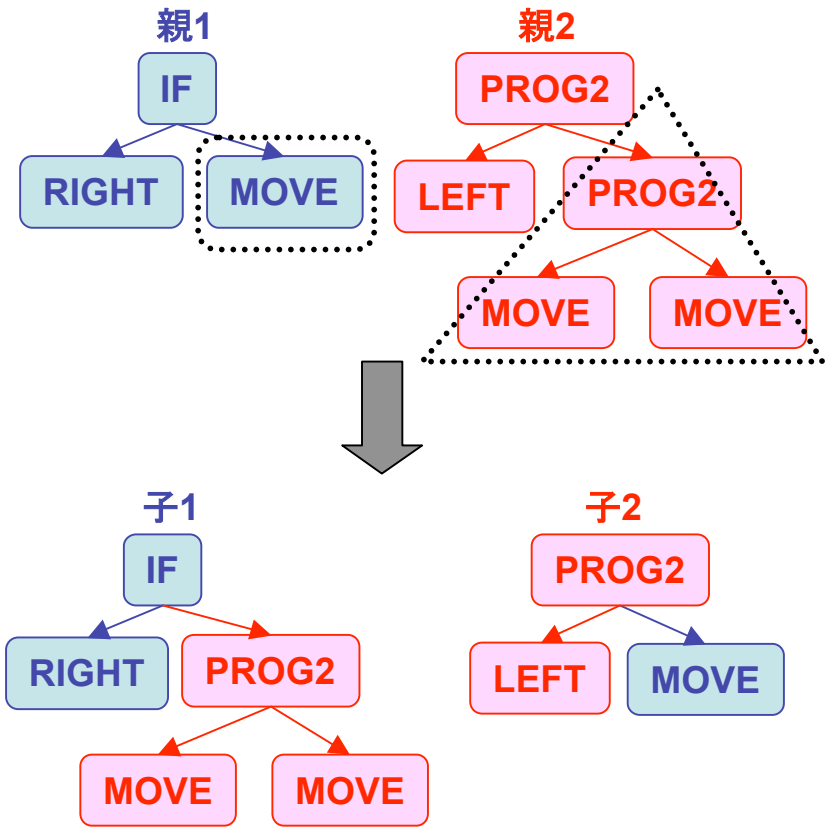
- 蟻の定義
- 終端記号(パラメータ無し)
  - MOVE: 前方に一歩進む
  - RIGHT: 右を向く
  - LEFT: 左を向く
- 非終端記号(パラメータ有り)
  - IF_FOOD_AHEAD(x,y): 餌があれば x, 無ければ y
  - Prog2(x,y): x を実行して y
  - Prog3(x,y,z)



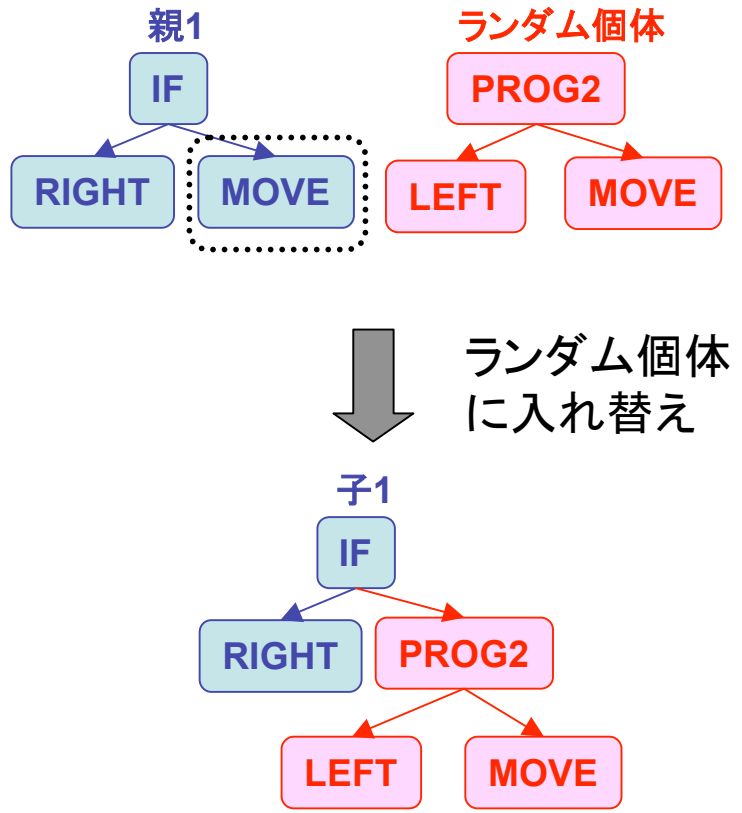
Q: 終端記号や非終端記号はどのように設計すべきか？

GPオペレータ(交叉・突然変異)

- 交叉

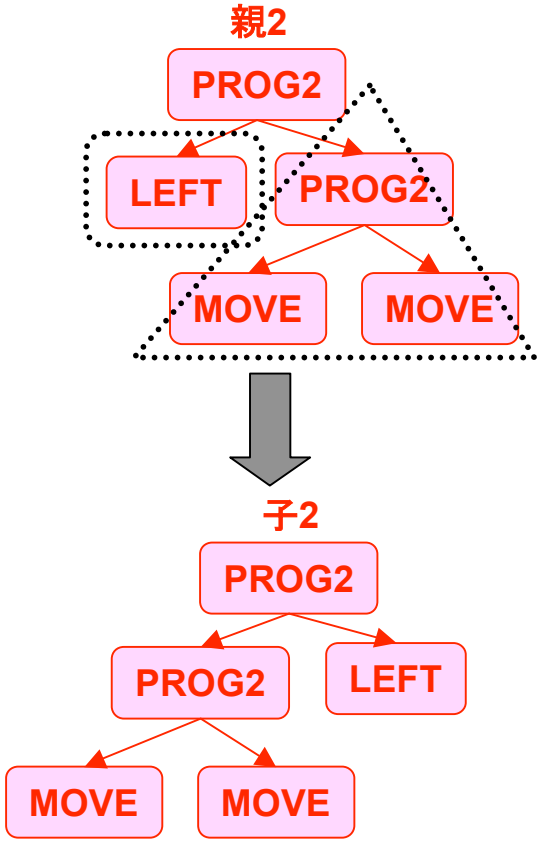


- 突然変異



GPオペレータ(逆位)

- 逆位



- 特徴
  - 利点
    - 自動プログラミング
  - 欠点
    - 個体サイズの増加
    - 個体の複雑化

Q: 解決方法は無いのか？

応用事例(GP)

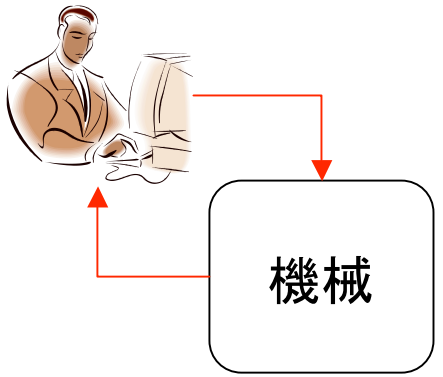
- 自動プログラム生成
  - LISP
- 同定問題
  - 関数同定
- ロボット制御
  - 行動ルール生成
  - 障害物回避
- 学習
- 推定

インタラクティブGA

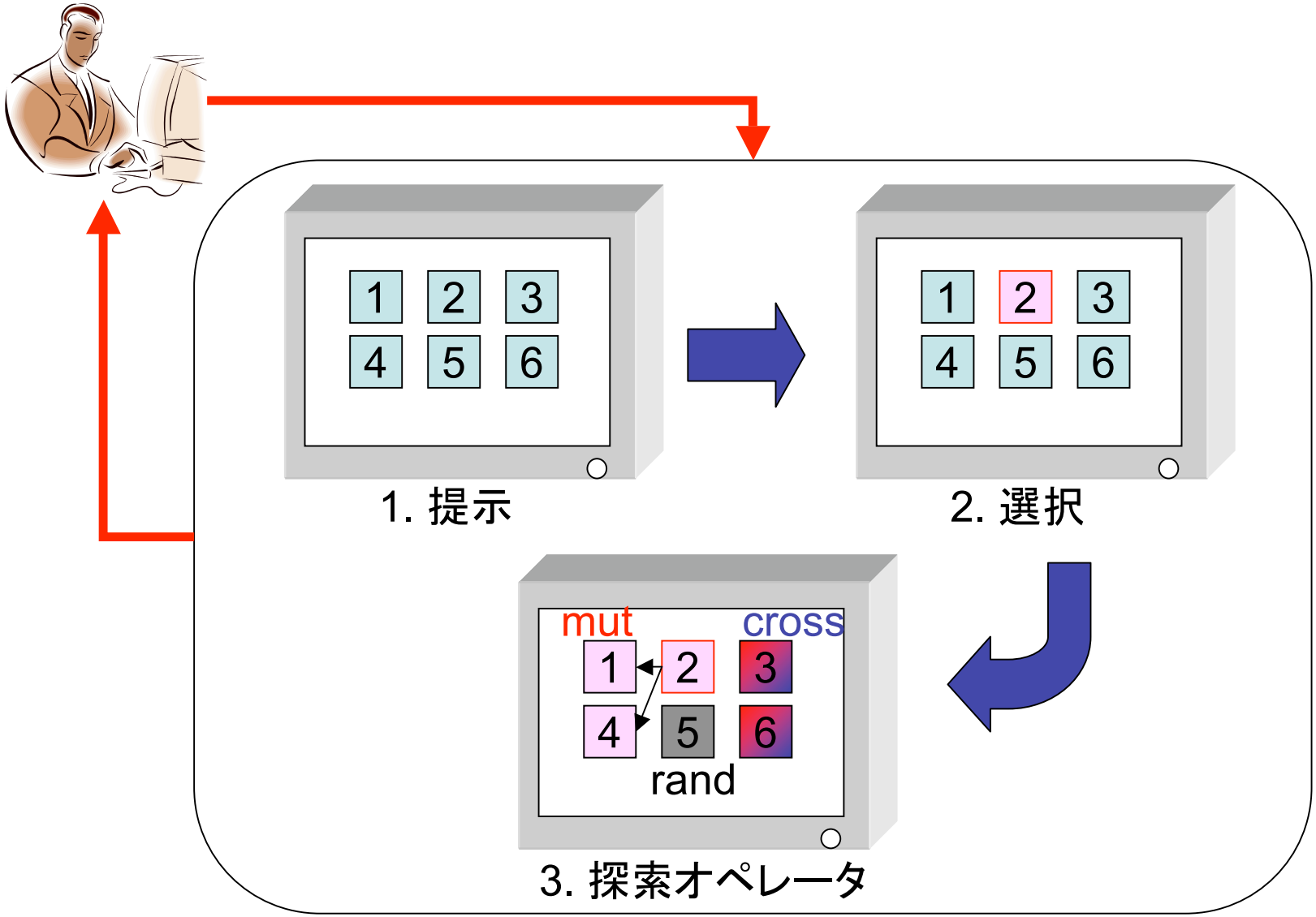
- 発想
  - 評価を人間に委ねる事により，定性的な評価が必要なアプリケーションへの適用.
- 利点
  - 主観/知識を利用した探索
  - 人間・コンピュータ間による協調作業
- 欠点
  - 世代数制約
  - 個体数制約
  - （利点でもある）評価軸の変化

Q: 人間の感性を含めた事の利点・欠点は何だろう？

- その他の特徴
  - インタラクティブなやりとり
    - 人間の利点と欠点
    - 機械の利点と欠点
  - を補うように設計するとベター



システム面からの特徴:GUI



応用事例(インタラクティブGA)

- 音/画像等の自動生成・生成アシスト
  - Biomorph(http://www2d.biglobe.ne.jp/~aquila/alife/a3/a3.html)
  - 自然画像データを入力とした紅型風画像生成ツールの開発(嘉数@遠藤研H16)

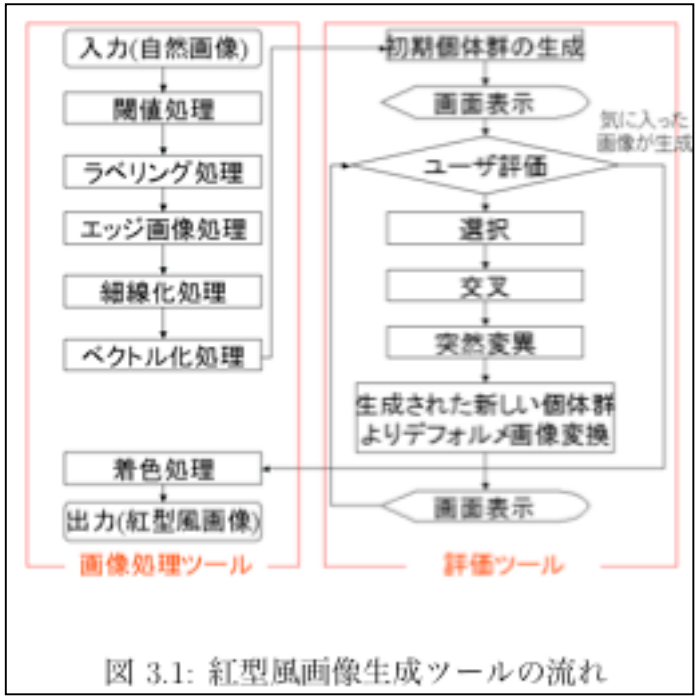
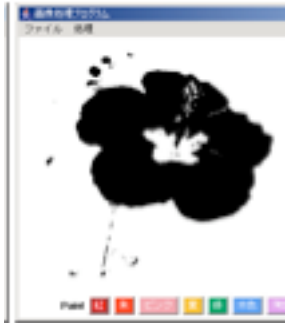
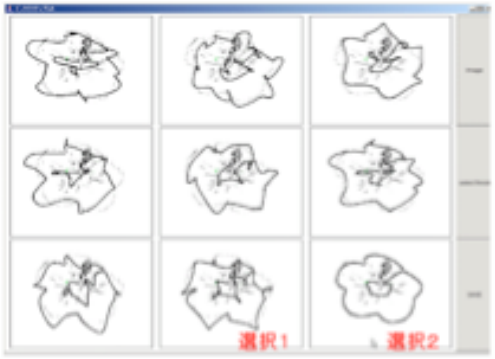


図 3.1: 紅型風画像生成ツールの流れ

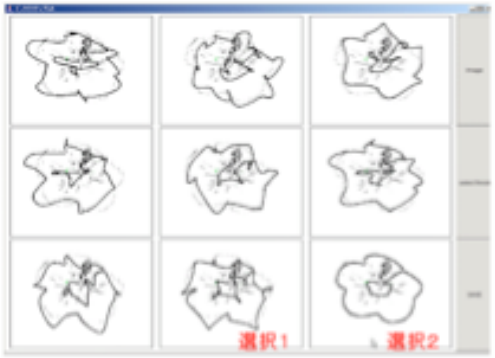


紅型風画像生成ツール

初期世代

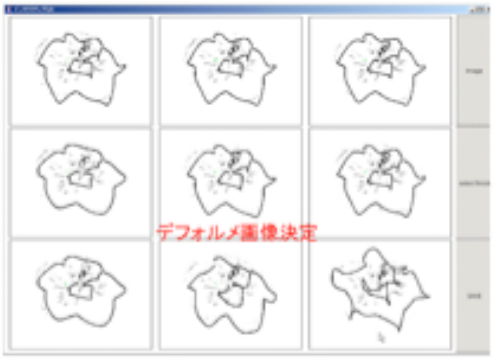


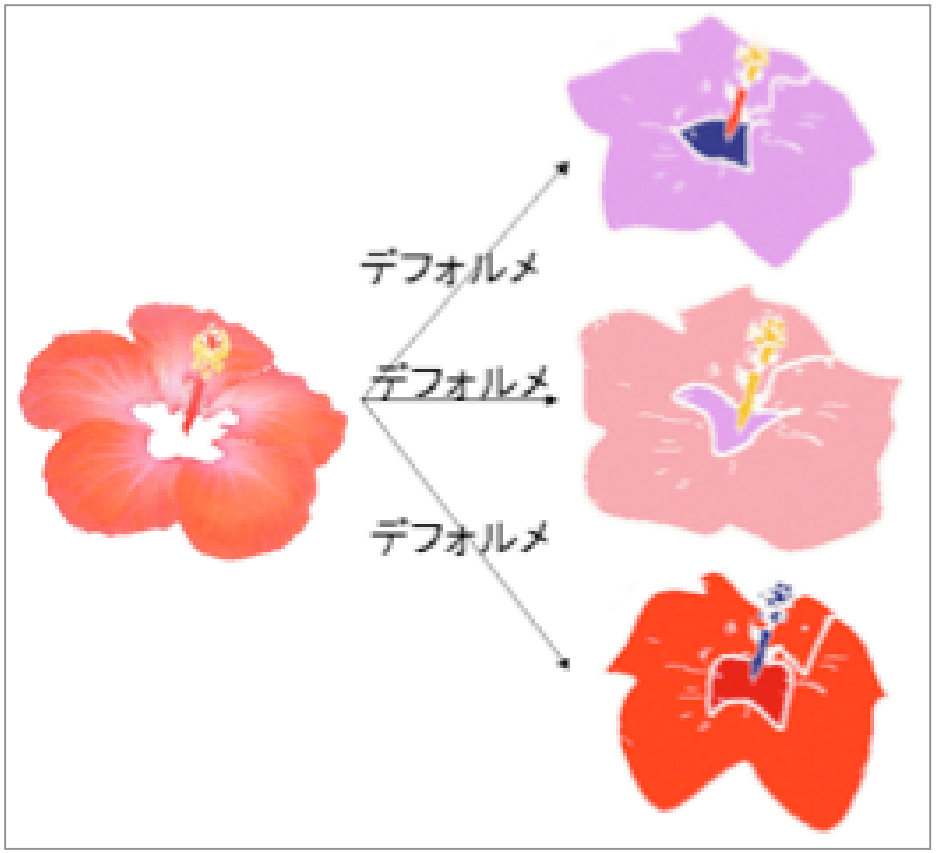
1世代



...

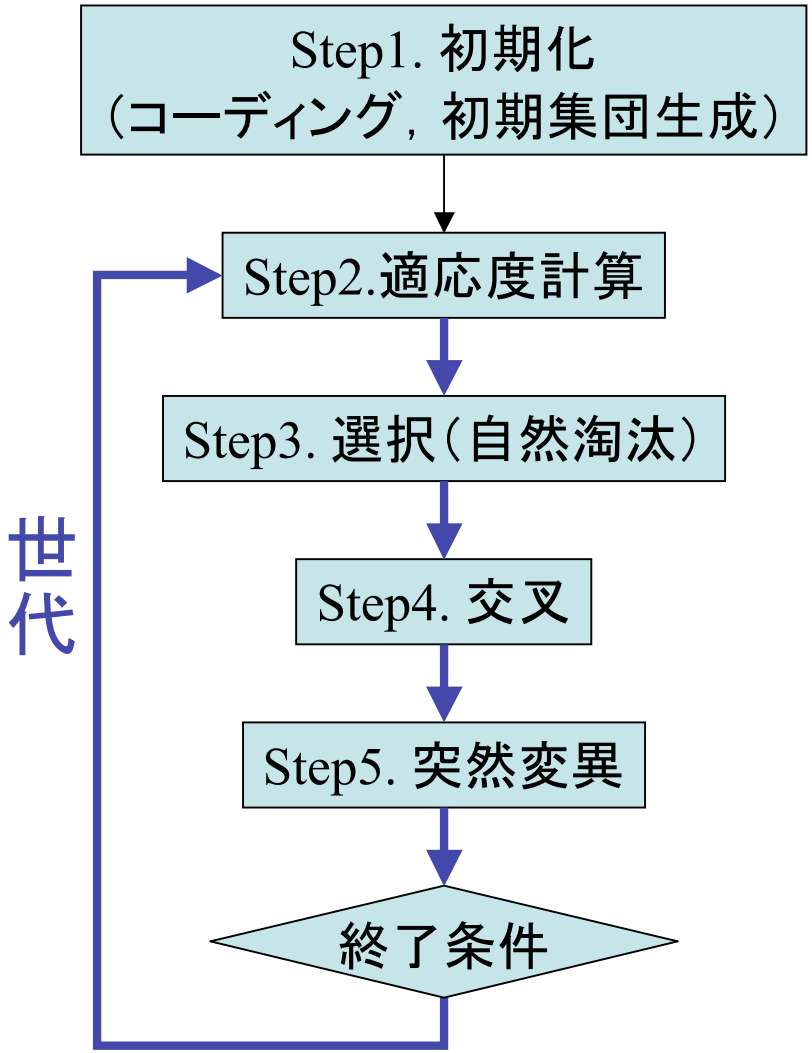
9世代





2. コードから読むGA

GAの処理手順とコーディング例



- GA for Knapsack
  - http://www.eva.ie.u-ryukyu.ac.jp/~tnal/Job/GA/Readme.html
  - を基に，設計方針，コーディング例を読み進める

入力関係(方針)

- 問題:コマンドライン上でファイル名を指定
- GAパラメータ:ヘッダーファイルにて指定
  - 例外)乱数シードはコマンドライン上で指定

prompt> **run_ga.sh 1 1 KP_List.data 100**

問題＋パラメータ
1)初期集団作成用シード値(下記の例では1),
2)GAオペレータ用シード値(同様に1),
3)荷重ファイル(同様にKP_List.data),
4)終了世代数(同様に10)

シェルスクリプト:run_ga.sh
指定パラメータ・問題に基づき，探索を実行

10
20
30
40
50
90
80
70
60
55
15
28
34
89
38

入力関係（コード例）

- 実行時の引数読み込み: ga.c

赤字は悪い例

```
/* global variables */
/* for Knapsack Problem */
int *item;          /* 加重ファイル */
char *itemfile;   /* ファイル名 */
```

```
main(int argc, char **argv)
{
    int seed_pop;   /* 集団用シード値 */
    int seed_ga;   /* GAオペレータ用シード値 */

    /* メモリ確保 */
    item = (int *)malloc(sizeof(int)*ITEM_NUM);
    itemfile = (char *)malloc(sizeof(char)*1024);

    if( argc != 5 ){    /* 引数チェック */
        usage();   /* 使い方表示 */
    }else{
        /* 引数読み込み（ノーチェック） */
        seed_pop = atoi(argv[1]);
        seed_ga = atoi(argv[2]);
        itemfile = argv[3];
        max_generation = atoi(argv[4]);
    }
```

出力関係(方針)

- 結果ファイル(全結果)
  - 探索オペレータ適用毎に全個体を出力(動作確認用)
  - 各世代毎に最良個体を出力
- 全結果から切り出した結果
  - 世代毎の最良適応度の推移図(作成用データ)

```
>> generation no.0
>> max gene[1] =  1 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1
>> Fitness = 260
MAX 0 260
>> max_fitness = 260
generation no.1
# roulette_selection().
gene no.0 = 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0  => fitness = 200
gene no.1 = 1 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1  => fitness = 260
gene no.2 = 1 0 0 0 0 0 1 0 1 0 1 1 1 0 1 1 1 0 1 1  => fitness = 180
gene no.3 = 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0  => fitness = 120
gene no.4 = 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0  => fitness = 220
gene no.5 = 1 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1  => fitness = 200
gene no.6 = 0 0 1 0 0 1 1 0 1 1 1 0 0 1 0 1 0 1 0 0  => fitness = 160
gene no.7 = 1 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1  => fitness = 240
gene no.8 = 1 0 0 0 0 0 1 0 1 0 1 1 1 0 1 1 1 0 1 1  => fitness = 120
gene no.9 = 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0  => fitness = 240
# singlepoint_crossover().
gene no.0 = 0 0 1 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1  => fitness = 200
gene no.1 = 1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 1 0 0 1 0  => fitness = 260
```

全結果

最良適応度推移

1	260
2	220
3	220
4	280
5	260
6	280
7	280
8	280
9	280
10	260

出力関係（コード例）

- run_ga.sh
  - GAを実行し，標準出力結果をファイルにリダイレクト
  - 最良適応度を切り出しやすく出力

```
#! /bin/sh

# For ga.c.

$1 $2 $3 $4 $5 > $2-$3-$4-$5

echo 'grep ">> max_fitness" stdout > stdout.max'
grep ">> max_fitness" $2-$3-$4-$5 | tr -s " " " " | cut -f4 -d" " | cat -n > $2-$3-$4-$5.max

echo 'tail stdout > stdout.end'
tail $2-$3-$4-$5 > $2-$3-$4-$5.end
```

run_ga.sh 1 1 KP_List.data 100 > 1-1-KO_List.data-100

集団生成

```
main(){
    /* initialize population */
    initialize_pop_binary(seed_pop);
}
```

```
extern int gene[POP_SIZE][GENE_LENGTH];

void initialize_pop_binary(int seed)
{
    int i;

    srand(seed);
    for(i=0; i<POP_SIZE; i++){
        /* gene作成 */
        initialize_gene_binary(gene[i]);
    }

}   /* end of initialize_pop_binary() */
```

集団生成

```
/*
  ### GA用バイナリストリングを一つ作成
  */
void initialize_gene_binary(int *str)
{
    int i;

    /* make binary string */
    for(i=0; i<GENE_LENGTH; i++){
        *(str+i) = random_i(2);
    }

}   /* end of nitialize_gene_binary() */
```

個体生成

選択（ルーレット保存）

```
void roulette_selection()
{
    static int sum_of_fitness; /* 適応度総和 */
    static double border; /* 境界 */
    static double r;
    static int i,j;
    static int num;
    int new_gene[POP_SIZE][GENE_LENGTH];
    int new_fitness[POP_SIZE];

    sum_of_fitness = 0;
```

前準備

```
    /* 総和計算 */
    for (i=0; i<POP_SIZE; i++)
        sum_of_fitness = sum_of_fitness + fitness[i];
```

選択操作

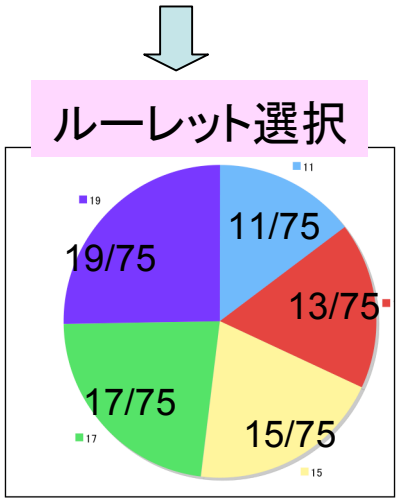
```
    for (i=0; i<POP_SIZE; i++){
        /* ルーレットを回し，ダーツを投げる */
        r = sum_of_fitness * (random_i(10001)/(10000.0));
        num = 0;
        border = fitness[0];

        while(border<r){ /* ダーツのあたった箇所を検索 */
            num++;
            border = border + fitness[num];
        }
        /* あたった箇所を増殖 */
        for (j=0;j<GENE_LENGTH;j++)
            new_gene[i][j] = gene[num][j];
        new_fitness[i] = fitness[num];
    }
```

```
    for(i=0; i<POP_SIZE; i++){
        for(j=0; j<GENE_LENGTH; j++)
            gene[i][j] = new_gene[i][j];
        fitness[i] = new_fitness[i];
    }
}
```

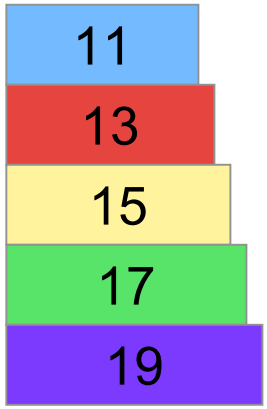
実際の集団に適用（コピー）

ルーレット選択のイメージ

e.g.) S1:11点, S2:13点, S3:15点, S4:17点, S5:19点



1次元表現



総和

=75

1〜75の乱数を生成

1〜11なら → 11 を選択
12〜24なら → 13 を選択
25〜39なら → 15 を選択
40〜56なら → 17 を選択
57〜75なら → 19 を選択

一点交叉

```
void singlepoint_crossover()
{
    int gene1[GENE_LENGTH];
    int gene2[GENE_LENGTH];
    unsigned char work;
    int         i,j,k;
    int         c_pos, c_pos2;
    double      r;

    for(i=0; i<(POP_SIZE-1); i=i+2){
        /* 指定された確率に基づいて実行 */
        r = random_i(10001)/(10000.0);
        if( r <= CROS_RATE ){

            for(j=0; j<GENE_LENGTH; j++){
                gene1[j] = gene[i][j];
                gene2[j] = gene[i+1][j];
            }

            /* 乱数を持ちいて交叉位置を決定し、
               その値をc_posへ代入する */
            c_pos = 0;
            c_pos = random_i(GENE_LENGTH);

            /* 交叉 */
            for(j=c_pos;j<GENE_LENGTH;j++){
                work = gene1[j];
                gene1[j] = gene2[j];
                gene2[j] = work;
            }

            /* geneにコピー */
            for(k=0; k<GENE_LENGTH; k++){
                gene[i][k] = gene1[k];
                gene[i+1][k] = gene2[k];
            }
        }  /* end of if(c_rate) */

    }  /* end of for(i) */

}  /* end of singlepoint_crossover() */
```

突然変異

```
void mutation_binary()
{
    int i, j;
    double r;
    int work;   /* 作業用変数 */
    int pos;

    for(i=0; i<POP_SIZE; i++)
        for(j=0; j<GENE_LENGTH; j++){
            /* 指定された確率に基づいて実行 */
            r = random_i(10001)/(10000.0);
            if( MUTE_RATE > r ){
                pos = random_i(GENE_LENGTH);

                /* posを対立遺伝子に交換 */
                if( gene[i][pos] == 0 ){
                    gene[i][pos] = 1;
                }else{
                    gene[i][pos] = 0;
                }
            }
        }   /* end of for(j,GENE_LENGTH) */
}
```

Week4 以降に向けて:
どんなシミュレータを作成するか？

- 対象問題から考える
  - 組み合わせ最適化問題
  - 人工生命関連
    - Karl Sims: http://www.genarts.com/karl/
    - Thomas S. Ray: http://www.genarts.com/karl/
    - 人工生命の宝庫: http://www2.create.human.nagoya-u.ac.jp/~ari/stuff/alifesoft.html
- シミュレータの利用形態から考える
  - GUI / インタフェース
  - 教材用
  - プレゼンテーション用
- その他ネット/文献サーベイ
- 等，各々に応じた実現方法を検討し，ベターな実装を選択する.

まとめ

- 拡張GA
  - GP
  - インタラクティブGA
- コードから読むGA
  - ナップサック問題
- システム開発に向けた準備

- 課題: Level 3
  - GAで解く問題を探し出し，定式化せよ．なお，翌週以降ではこれをもとに要求仕様・要件定義・外部設計・内部設計・テスト計画を立案していく事になる．それを踏まえた上で，見つけて来た問題に付いては十二分にサーベイをすること．（要求仕様まで考えておくと，今後のスケジュールがスムーズになります！）
- オプション
  - その際，その問題ならではの特性をまとめ，それをうまくGAで処理するための方法論について考察せよ

次回

- 4-6週目：企画
  - 問題定義（要求仕様，要件定義）
  - 外部設計
  - 内部設計
  - テスト計画
- 注意
  - 今後は進捗具合の報告とそれに対する指導がメインとなる．
  - 残りの約3ヶ月間をどのように使うかのスケジュール調整にも留意する事（e.g., UNIX実験との兼ね合い等）