アルゴリズム論

平成22年11月22日

Genetic Algorithm [1]

課題 last

- ［課題 last］NP 困難な最適化問題を一つ選び，その近似解を遺伝アルゴリズムを用いて求めるプログラムを実現せよ。
  染色体表現は，二進数列・順列のいずれのモデルでもよい。
  プログラムを示すだけでなく，問題の記述と適応度計算関数 fitness の解説は必須である。
  probX, probM, Npop, Times などのパラメータを適切に調整する過程の実験データが提示されることを望む。

遺伝(的)アルゴリズム

- Genetic Algorithm (GA)
  - 最適化問題の汎用的近似解法 meta-heuristics
  - 複数の実行可能解を保持
  - 解を生物の進化のアナロジーを用いて改善

- 近似解法 ⇔ 生物進化 [対比の例]
  - 実行可能解 ⇔ 染色体(遺伝子の列)
  - 第 t 反復の解の集合 ⇔ 第 t 世代の個体集団
  - 新たな解の生成 ⇔ 交叉／突然変異
  - 実行不可能な解 ⇔ 致死遺伝子(をもつ染色体)
  - 目的関数(最大化) ⇔ 適応度

染色体は遺伝子の並び

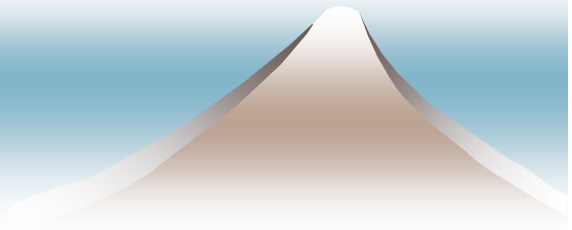
- 染色体表現
  - 二進数列：0 と 1 の並び
    … 遺伝アルゴリズム研究は二進数列から始まった
  - 三進数列：例えばＧとＣとＰ[ジャンケンゲーム]
  - 順列：重複しない整数値の並び
    … 古典的な GA と区別して，進化アルゴリズム
    EA: evolutionary algorithm と呼ばれることもある

- 染色体の長さは，通常は，一定
  [可変長の染色体を扱った研究もある]

染色体の例

- 長さが５の場合の例：
  - 二進数列　01100，11001，00000，10111
  - 三進数列　GGCPC，CCCCC，CGGPC，GCPGC
  - 順列　　23514，12345，54213，32541

- 長さが n の二進数列は：2^n 個ある
- 長さが n の三進数列は：3^n 個ある
- 長さが n の順列は：$n!$ 個ある



染色体は一つの解

- 各染色体：入力した具体例の一つの解（に対応）
- （実行）可能解：問題の条件を満足する解
  条件を満足しない解を導く染色体は
  ″致死遺伝子„ をもつという
- 評価値／目的関数／適応度
  各可能解の良さを表す値
- 順列そのものが可能解であれば話は簡単
- 通常は，順列を ″よりどころ„ にして，
  入力した具体例の可能解を作成する

permutation-encoding and greedy-decoding

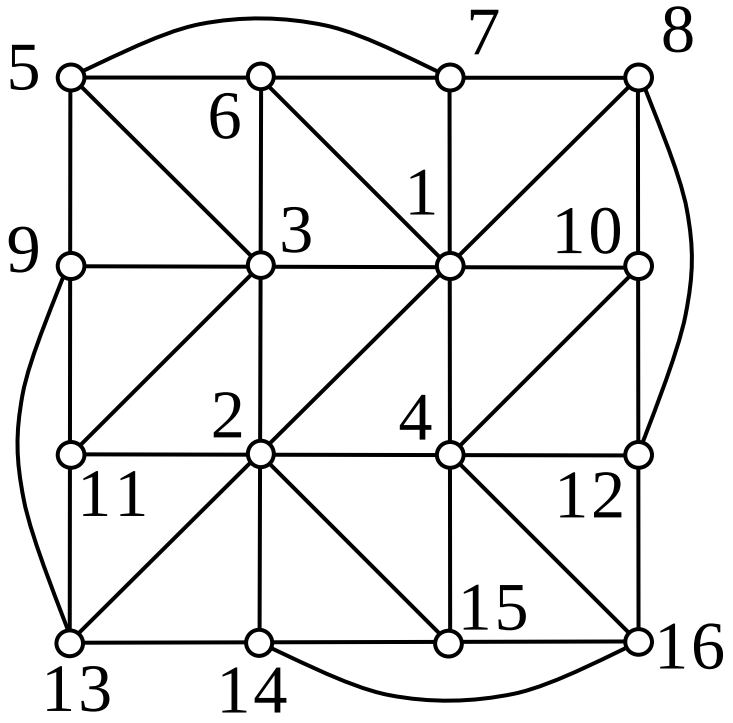
greedy-decoding［例：彩色問題］

- 例としてグラフの彩色問題を考える
- 頂点の順列（染色体）が与えられたとき，
  その順列を〝よりどころ〟として，
  できるだけ少ない色数でグラフを彩色する
- 使われた色数がその順列（染色体）の評価値となる
- 順列から可能解への変換：
  　できるだけ良い可能解を得たい
  　余り計算時間はかけたくない

greedy-decoding［１］

順列：$\alpha = (5,6,7,3,13,14,2,12,16,15,10,9,1,4,8,11)$

greedy = 貪欲な，
欲張りの，
がつがつした

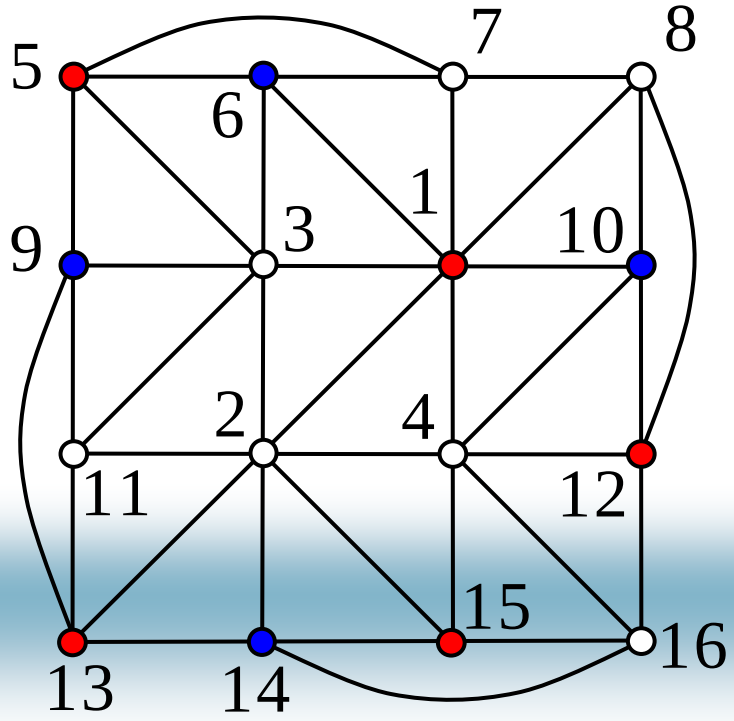


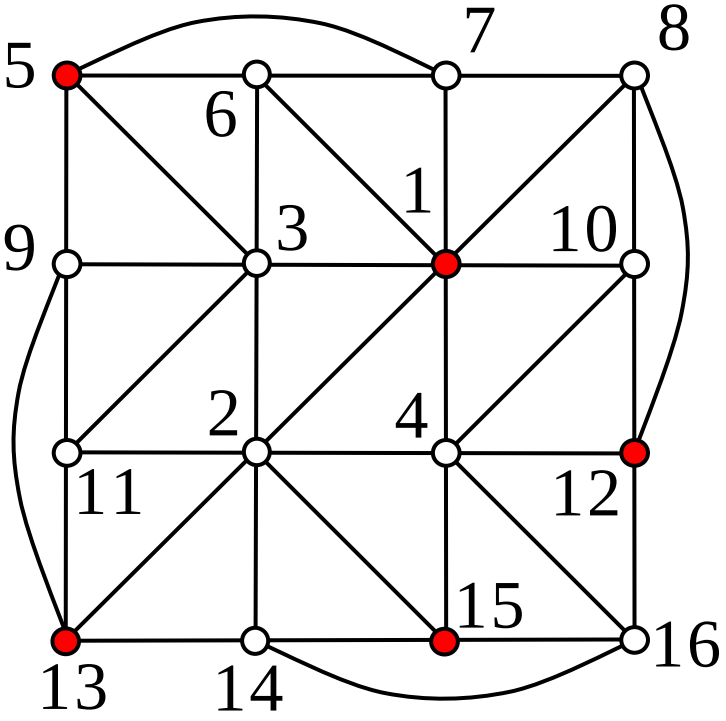
まず頂点 5 に色１を塗る
以下，順列に現れる順に頂点を調べ
色１を塗れる頂点を彩色する

greedy-decoding［２］

順列：$\alpha = (5,6,7,3,13,14,2,12,16,15,10,9,1,4,8,11)$

次にまだ無色の頂点を
先頭から調べ，色２を塗る

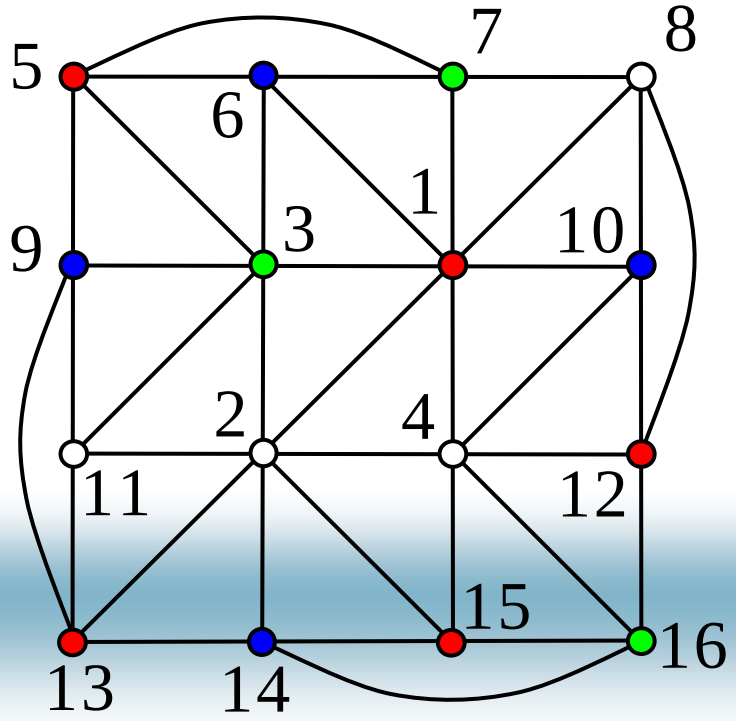


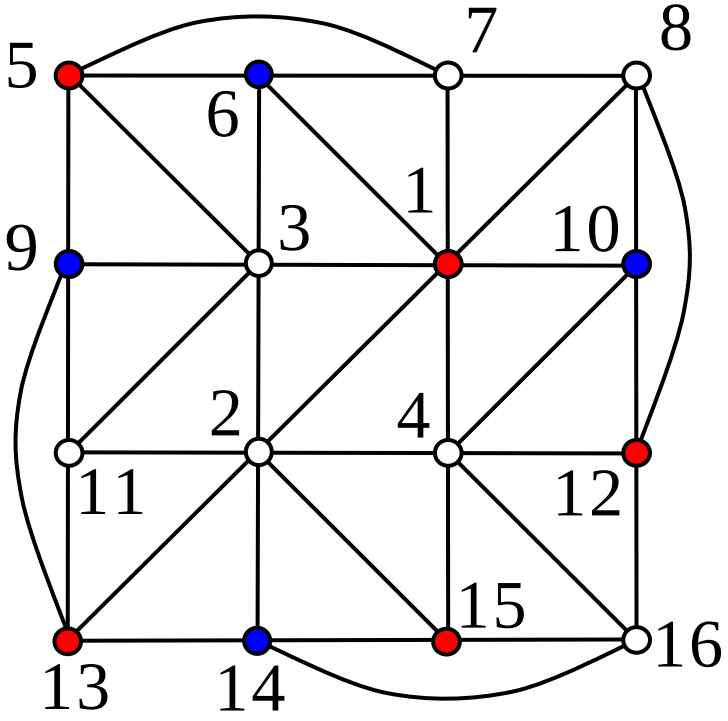


greedy-decoding［３］

順列：$\alpha = (5,6,7,3,13,14,2,12,16,15,10,9,1,4,8,11)$

次にまだ無色の頂点を
先頭から調べ，色３を塗る

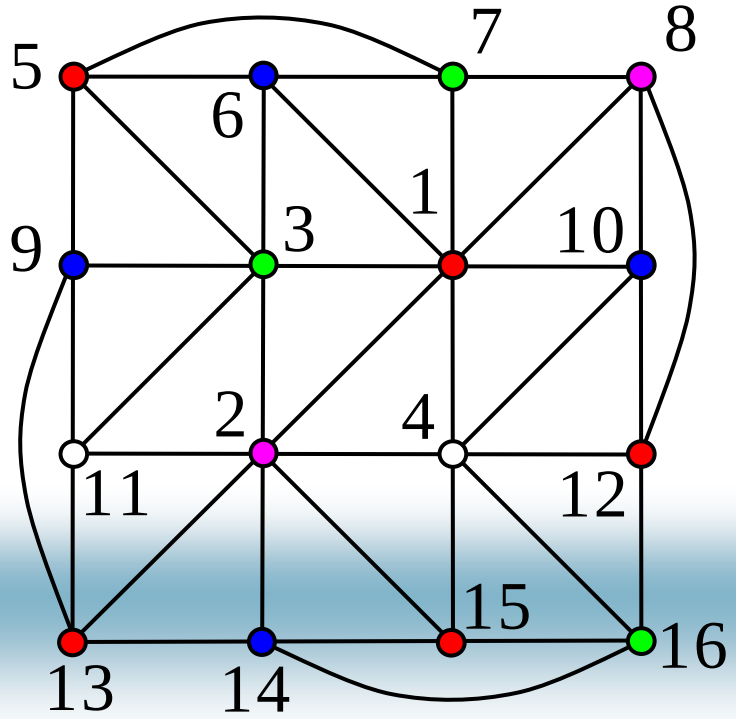


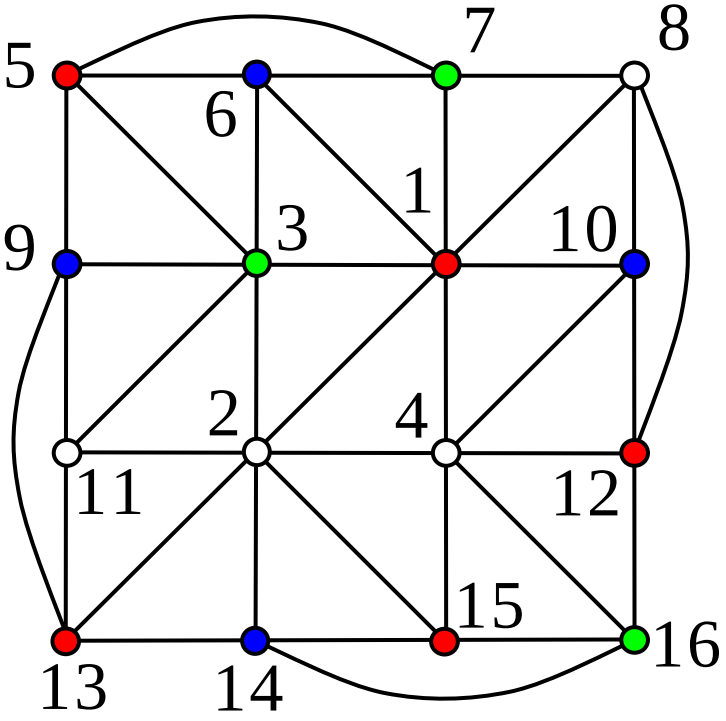


greedy-decoding［４］

順列：$\alpha = (5,6,7,3,13,14,2,12,16,15,10,9,1,4,8,11)$

次にまだ無色の頂点を
先頭から調べ，色４を塗る



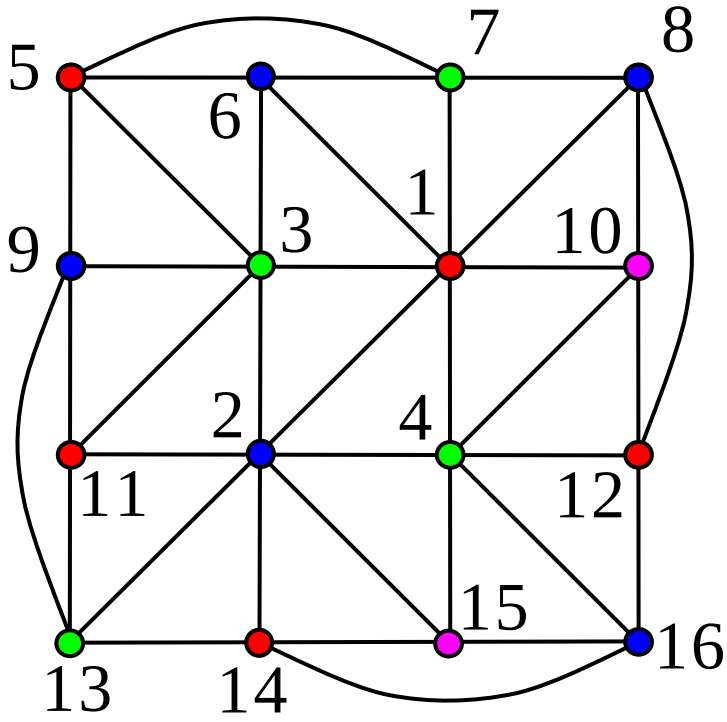


残った頂点 4 と 11 に色５

α の評価値は５

greedy-decoding［５］

greedy-decoding の条件は
最適解を導く順列が存在すること



このグラフの彩色数は４

例えば次の順列 β がこの彩色を導く

$$\beta = (1,5,11,12,14,2,6,8,9,16,3,4,7,13,10,15)$$

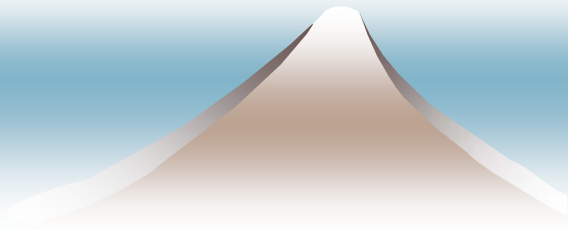
GA の一般的な流れ

```
def GeneticAlgorithm(…):
  … 個別問題の設定 …    # Length：染色体の長さ
  … GA パラメータの設定 …
                        # Npop：個体数,  Times：  終了世代数
                        # probX：交叉確率, probM：突然変異確率
  group = initialize(Length, Npop)    # 初期集団生成
  for k in range(Times):
    evaluate(group, … 個別問題 …)   # 適応度の評価
    eliteChr = elite(group)           # エリートを記憶
    prntAll(group, k)
    group = roulette(group)           # 次世代をルーレット選択
    crossover(group, probX)           # 交叉
    mutation(group, probM)            # 突然変異
    elitePrs(group, eliteChr, 2)      # エリートを２個保存する
  … 結果の出力 …
  return
```

パラメータを読む

```
def parameter(paras = ""):
  """ パラメータを読む """
  Times, Npop, Seed = (100, 25, 109)  # default値
  if paras = "":
    paras = []
  else:
    paras = paras.split("-")
  for p in paras:
    if   p[0] == "t":  Times = int(p[1:])
    elif p[0] == "p":  Npop  = int(p[1:])
    elif p[0] == "s":  Seed  = int(p[1:])
  return (Times, Npop, Seed)

# paras の例："t200-p30-s29"  "p50"  "s91-t200"
# 実際はパラメータの数はもっと増える
```



初期集団生成

```
import random    # 乱数に関するモジュール

def initialize(Length, Npop):
  """ 初期集団生成：ランダムな順列を Npop 個 """
  group = []
  for k in range(Npop):
    chromo = range(Length)    # chromo = [0, 1, ..., Length-1]
    random.shuffle(chromo)    # chromo = [3, 9, ..., 1, 0, 5]
    chromo.append(0)          # chromo[-1] には適応度を入れる
    group.append(chromo)
  return group
```

chromo が [2, 4, 0, 1, 3, 25] なら，Length = 5
　染色体は (2,4,0,1,3) であり，その適応度(評価値) が 25

生物集団を出力

```
def prntAll(group, time):
  """ 全染色体を出力 """
  print "%4d - " % time,
  for chromo in group:
    # print "{%d} %s" % (chromo[-1], chromo[:-1])
    print chromo[-1],
  print
  return

# list[:-1] ≡ list[0] ～ list[-2] ≡ list[0:-1]
```

適応度の評価

```
def evaluate(group, adjL):
  for chromo in group:
    fitness(chromo, adjL)
  return


def fitness(chromo, adjL):    # 問題ごとに作成
   ...
   ...
  chromo[-1] = *****
  return
```

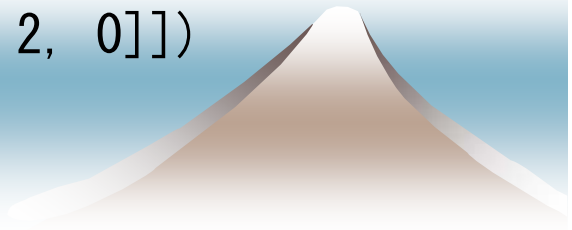
エリートを記憶

```
def elite(group):
  """ 生物集団 group 中のエリートを返す """
  largestPos, largestVal = (0, group[0][-1])
  for k in range(1, len(group)):
    if largestVal < group[k][-1]:
      largestPos, largestVal = (k, group[k][-1])
  return group[largestPos][:]
```

◎ 戻り値を group[largestPos][:] としておけば,
group[largestPos][0] ～ group[largestPos][-1] までの値を
要素とするリストそのものが戻される。

リストの不思議

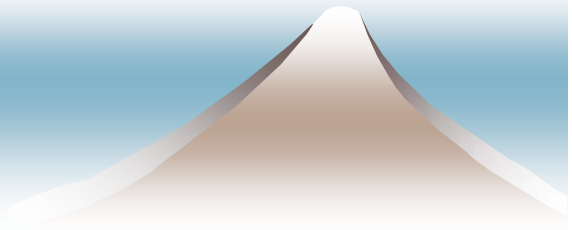
```
>>> A = initialize(4, 2)
>>> A
[[0, 2, 3, 1, 0], [3, 0, 1, 2, 0]]
>>> top = A[0]
>>> top, A
([0, 2, 3, 1, 0], [[0, 2, 3, 1, 0], [3, 0, 1, 2, 0]])
>>> A[0][0] = 9
>>> top, A
([9, 2, 3, 1, 0], [[9, 2, 3, 1, 0], [3, 0, 1, 2, 0]])
>>> top = A[0][:]
>>> top, A
([9, 2, 3, 1, 0], [[9, 2, 3, 1, 0], [3, 0, 1, 2, 0]])
>>> A[0][0] = 99
>>> top, A
([9, 2, 3, 1, 0], [[99, 2, 3, 1, 0], [3, 0, 1, 2, 0]])
```



ルーレット選択

```
def roulette(oldGrp):
  """ ルーレット選択 """
  strip = [oldGrp[0][-1]]              # 要素１つのリスト
  for k in range(len(oldGrp) - 1):
    strip.append(strip[-1] + oldGrp[k + 1][-1])
                # 布切れの最後に k + 1 番目の評価値の布を加える
  newGrp = []
  while len(newGrp) < len(oldGrp):
    newGrp.append(oldGrp[rotate(strip)][:])
  return newGrp
```

適応度: 4, 5, 21, 11, 2, ...
strip: 4, 9, 30:(41)

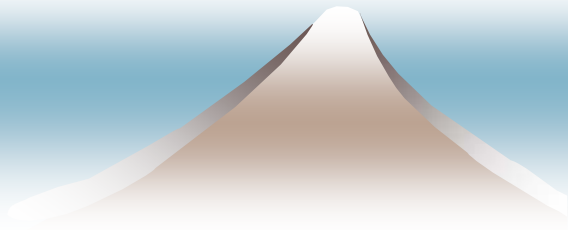


ルーレットを回す

```
def rotate(strip):
  """ ルーレットを回す """
  value = random.random() * strip[-1]
  left, right = (0, len(strip) - 1)
  while left < right:
    middle = (left + right) / 2
    if strip[middle] < value:
      left = middle + 1
    else:
      right = middle
  return right
```

交叉

```
def crossover(group, probX):
  """ 交叉確率 probX """
  genes = group[0][:-1]      # 遺伝子の集合，順序はランダム
  for k in range(0, len(group), 2):
    if random.random() > probX:   # 0 <= random() < 1
      continue
    pos2 = random.sample(genes, 2)
    posL = range(min(pos2), max(pos2))
    group[k]     = PMXbasic(group[k], group[k + 1], posL)
    group[k + 1] = PMXbasic(group[k + 1], group[k], posL)
  return
```



交叉：crossover

◎ 二つの染色体の遺伝子情報の一部を継承する新たな染色体を(通常は二つ)作る操作

◎ 染色体が順列表現の場合，工夫が必要

◎ 一点交叉：切断点 cut を指定する

left タイプ：遺伝子座 [:cut] に着目

rightタイプ：遺伝子座 [cut:] に着目

◎ 二点交叉：切断点 left と right を指定

in タイプ：遺伝子座 [left:right] に着目

outタイプ：遺伝子座 [:left] と [right:] に着目

二点交叉の切断点

$$\begin{array}{l|l|l} \alpha = \ 3 \ \ 2 & 4 \ \ 0 \ \ 5 & 7 \ \ 1 \ \ 6 \\ \hline \beta = \ 2 \ \ 0 & 3 \ \ 7 \ \ 6 & 1 \ \ 5 \ \ 4 \end{array}$$

◎ 染色体 α と β を二つの切断点で交叉させる説明図

◎ 染色体の長さは 8, α の遺伝子座 0 の遺伝子は 3, 遺伝子座 7 の遺伝子は 6

◎ 交叉範囲は, 遺伝子座 k が $2 \leqq k < 5$ の位置であり, 上の図では, left＝2, right＝5

◎ 交叉範囲をリストで示す : left＝2, right＝5

in タイプ : posL = [2, 3, 4]

out タイプ : posL = [0, 1, 5, 6, 7]

部分一致交叉：PMX

◎ partial mapped crossover：
部分写像交叉と呼ばれることもある

◎ 部分一致させる遺伝子座の範囲を選ぶ

◎ 一致させる遺伝子座に，相手側染色体の同一遺伝子座にある遺伝子が来るようにする

◎ 染色体が順列であるという条件を乱さないように交換する

PMXin の実行例

papa =	3	2	4	0	5	7	1	6
mama =	2	0	5	7	6	1	3	4

child =	3	2	4	0	5	7	1	6
	3	2	5	0	4	7	1	6
	3	2	5	7	4	0	1	6
	3	2	5	7	6	0	1	4

部分一致交叉

```
def PMXbasic(papa, mama, posL):
  """ partial mapped crossover """
  child = papa[:]
  for pos in posL:
    swap(child, pos, child.index(mama[pos]))
  return child

def swap(List, left, right):
  List[left], List[right] = (List[right], List[left])
  return
```

順序交叉：OX

◎ order crossover

◎ 一方の染色体の指定された範囲内の遺伝子はそのまま受け継ぎ，範囲外の遺伝子はもう一方の染色体に現れる順に並びかえる

OXin

papa =	3	2	4	0	5	7	1	6
mama =	2	0	5	7	6	1	3	4
papa の child =	7	2	4	0	5	6	1	3
mama の child =	3	0	5	7	6	2	4	1

◎ 順序交叉も４種類：

OXin, OXout, OXleft, OXright

突然変異

```
def mutation(group, probM):
  """ 突然変異確率 probM """
  genes = group[0][:-1]
  for chromo in group:
    if random.random() > probM:
      continue
    pos2 = random.sample(genes, 2)
    swap(chromo, pos2[0], pos2[1])
  return
```

GA の一般的な流れ

```
def GeneticAlgorithm(…):
  … 個別問題の設定 …    # Length：染色体の長さ
  … GA パラメータの設定 …
                        # Npop：個体数,  Times：  終了世代数
                        # probX：交叉確率, probM：突然変異確率
  group = initialize(Length, Npop)    # 初期集団生成
  for k in range(Times):
    evaluate(group, … 個別問題 …)   # 適応度の評価
    eliteChr = elite(group)           # エリートを記憶
    prntAll(group, k)
    group = roulette(group)           # 次世代をルーレット選択
    crossover(group, probX)           # 交叉
    mutation(group, probM)            # 突然変異
    elitePrs(group, eliteChr, 2)      # エリートを２個保存する
  … 結果の出力 …
  return
```

エリートの保存

```
def elitePrs(group, eliteChr, num):
  deleteS = random.sample(range(len(group)), num)
  for k in deleteS:
    group[k] = eliteChr[:]
  return
```

deleteS には, group の中の評価値の悪いものを num 個選ぶようにすることも考えられる。

課題 last

- ［課題 last］NP 困難な最適化問題を一つ選び，その近似解を遺伝アルゴリズムを用いて求めるプログラムを実現せよ。
  染色体表現は，二進数列・順列のいずれのモデルでもよい。
  プログラムを示すだけでなく，問題の記述と適応度計算関数 fitness の解説は必須である。
  probX, probM, Npop, Times などのパラメータを適切に調整する過程の実験データが提示されることを望む。