Genetic Algorithm Exercise

Genetic Algorithms for Single-objective Binary, Permutation, and Continuous Optimization

August 20th, 2017

1 Introduction

Single-objective optimization is the problem of finding a solution $\boldsymbol{x}$ that minimizes or maximizes an objective function f. Optimization problems can be roughly classified into combinatorial (discrete) optimization problems and continuous optimization problems. The class of combinatorial optimization problems can be further classified into binary and permutation optimization problems. This report presents seven programming exercises of genetic algorithms for single-objective binary, permutation, and continuous optimization. They are described in Section 2, 3 and 4, respectively.

You can learn how to implement genetic algorithms and the basic programming skills through experience. The exercises presented in this report do not require any expert programming skills. Also, you can implement genetic algorithms by any programming language (e.g., C, C++, Java, Matlab/Octave, Python, ...).

2 GA for Single objective binary optimization problems

2.1 One-max problem

The one-max problem is the most simple optimization problem. The solution of the one-max problem is a D-dimensional binary vector $\boldsymbol{x} = (x_1, ..., x_D)^{\mathrm{T}} \in \{0, 1\}^D$. The objective function $f : \mathbb{Z}^D \to \mathbb{Z}$ to be maximized is defined as follows:

$$f(\boldsymbol{x}) = \sum_{i=1}^{D} x_i \tag{1}$$

The number of variables whose values are 1 is the objective function value $f(\boldsymbol{x})$ of the solution $\boldsymbol{x}$. For example, if $\boldsymbol{x} = (1, 0, 1, 1, 0)^{\mathrm{T}}$, $f(\boldsymbol{x}) = 3$. As seen from equation (1), the optimal solution $\boldsymbol{x}^*$ is clearly $\boldsymbol{x}^* = (1, ..., 1)^{\mathrm{T}}$.

2.2 Exercise 1: Implement $(\mu + 1)$-GA

The goal of this exercise is to implement $(\mu + 1)$-GA. Traditionally, $(\mu + \lambda)$ notation is used in the evolutionary computation community, where μ and λ represent the population size and the number of children respectively. For example, $(123 + 456)$-GA indicates that the the population size and the number of children are 123 and 456 respectively. The $(\mu + 1)$-GA, also known as steady-state

GA, is a special case of $(\mu + \lambda)$-GA when $\lambda = 1$. Since $(\mu + 1)$-GA is easy to implement than $(\mu + \lambda)$-GA, it is suitable as the first exercise.

All variables used in this section are as follows:

- t: The number of generation (or iterations)
- n: The number of function evaluations. In general, $n \neq t$. However, $n = t$ only in the case of $(\mu + 1)$-GA
- $n^{\max}$: The maximum number of function evaluations
- μ: The population size
- λ: The number of children
- $\boldsymbol{P}$: The population. $\boldsymbol{P} = \{\boldsymbol{x}^1, ..., \boldsymbol{x}^\mu\}$
- $\boldsymbol{x}^i = (x_1^i, ..., x_D^i)^{\mathrm{T}}$: The i-th individual in the population $\boldsymbol{P}$
- $\boldsymbol{u} = (u_1, ..., u_D)^{\mathrm{T}}$: A child individual
- $\boldsymbol{x}^{\mathrm{bsf}}$: The best-so-far solution found during the search process
- $p_m \in [0, 1]$: Mutation rate. The most widely used setting is $p_m = 1/D$

Algorithm 1 shows the overall procedure of $(\mu + 1)$-GA. Algorithm 2, 3 and 4 also describe the procedure of the population initialization, uniform crossover, and bit-flip mutation, respectively. The search should be terminated when n reaches $n^{\max}$ or the optimal solution is found (in case of $f(\boldsymbol{x}^{\mathrm{bsf}}) = f(\boldsymbol{x}^*)$). If the $(\mu + 1)$-GA which you implemented can successfully find the optimal solution on the one-max problem instance with $D = 50$, this exercise is finished.

Algorithm 1: $(\mu + 1)$-GA

```
   //  Initialization phase
1  t ← 1;
2  Initialize the population P = {x^1, ..., x^μ} randomly (Algorithm 2);
3  Evaluate each individual x^i (i ∈ {1, ..., μ}) in P by equation (1);
4  while The termination criteria are not met do
       //  Step 1.  Mating selection
5      Randomly select two parent individuals x^a and x^b from P so that a ≠ b;
       //  Step 2.  Variation operator1:  Crossover
6      Generate a child u by applying uniform crossover (Algorithm 3) to x^a and x^b;
       //  Step 3.  Variation operator2:  Mutation
7      Apply bit-flip mutation (Algorithm 4) with the mutation probability p_m = 1/D to u;
       //  Step 4.  Evaluate the child u
8      Evaluate the child u by equation (1);
9      n ← n + 1;
       //  Step 5.  Update the best-so-far solution x^bsf
10     if f(u) > f(x^bsf) then
11         x^bsf ← u;
       //  Step 6.  Environmental selection
12     Randomly select an individual x^c from P;
13     if f(u) ≥ f(x^c) then
14         x^c ← u;
15     t ← t + 1;
16 Show the best-so-far solution x^bsf and its objective function value f(x^bsf) to an user (you);
```

Algorithm 2: The initialization procedure of the population

```
   //  For each individual x^i in the population P = {x^1, ..., x^μ}
1  for i ∈ {1, ..., μ} do
       //  For each decision variable x_j in the solution x = (x_1, ..., x_D)^T
2      for j ∈ {1, ..., D} do
3          if rand[0, 1] < 0.5 then
4              x^i_j = 0;
5          else
6              x^i_j = 1;
```

Algorithm 3: Uniform crossover

```
1  for j ∈ {1, ..., D} do
2      if rand[0, 1] < 0.5 then
3          u^i_j = x^a_j;
4      else
5          u^i_j = x^b_j;
```

Algorithm 4: Bit-flip mutation

```
for j ∈ {1, ..., D} do
    if rand[0, 1] < p_m then
        if u_j = 0 then
            u_j = 1;
        else
            u_j = 0;
```

Algorithm 5: Binary tournament selection without replacement for the selection of parent individuals $\boldsymbol{x}^a$ and $\boldsymbol{x}^b$

```
1 Randomly select four individuals x^d, x^e, x^f and x^g from the population P so that
  d ≠ e ≠ f ≠ g;
  //  Select a parent individual x^a
2 if f(x^d) > f(x^e) then
3 |  x^a ← x^d;
4 else
5 |  x^a ← x^e;
  //  Select another parent individual x^b
6 if f(x^f) > f(x^g) then
7 |  x^b ← x^f;
8 else
9 |  x^b ← x^g;
```

2.3 Exercise 2: Change the environmental selection method in Algorithm 1

In Algorithm 1, an individual $\boldsymbol{x}^c$, which is possibly replaced by a child $\boldsymbol{u}$, is uniformly randomly selected (line 12). What happens when the environmental selection method is changed as follows?

- Select the worst individual $\boldsymbol{x}^{\text{worst}}$ in $\boldsymbol{P}$ and $\boldsymbol{x}^c = \boldsymbol{x}^{\text{worst}}$

2.4 Exercise 3: Change the mating selection method in Algorithm 1

In Algorithm 1, parent individuals $\boldsymbol{x}^a$ and $\boldsymbol{x}^b$ are randomly selected from $\boldsymbol{P}$ (line 5). What happens when the mating selection method is modified to binary tournament selection described in Algorithm 5?

2.5 Exercise 4: Evaluate the performance of the GA on the order-3 deceptive problem

In principle, the GA which you implemented for the one-max problem can be applied any binary optimization problems by changing the objective function (i.e., equation (1)). The goal of this exercise is to evaluate the performance of the GA on the order-3 deceptive problem. While the landscape of the onemeax problem is unimodal, that of the order-3 deceptive problem is multimodal. A three-dimensional order-3 deceptive problem (i.e., $\boldsymbol{x} = (x_1, x_2, x_3)^{\mathrm{T}}$) is defined as follows [3]:

$$
\begin{aligned}
&f(1,1,1) = 30, \\
&f(1,1,0) = 0, f(1,0,1) = 0, f(0,1,1) = 0, \\
&f(1,0,0) = 14, f(0,1,0) = 22, f(0,0,1) = 26, \\
&f(0,0,0) = 28
\end{aligned}
\tag{2}
$$

The optimal and sub-optimal solutions of this problem are $\boldsymbol{x}^* = (1,1,1)^{\mathrm{T}}$ and $\boldsymbol{x}^{\text{sub}*} = (0,0,0)^{\mathrm{T}}$, respectively.

Algorithm 6: The procedure of calculating the objective function value $f(\boldsymbol{x})$ of the order-3 deceptive problem

```
s ← 0;
j ← 1;
while j < D do
    x^tmp ← (x_j, x_{j+1}, x_{j+2});
    s ← s + f(x^tmp) (see equation (2));
    j ← j + 3;
f(x) ← s;
```

A D-dimensional order-3 deceptive problem (i.e., $\boldsymbol{x} = (x_1, ..., x_D)^{\mathrm{T}}$) consists of D three-dimensional problems defined in (2), where D must be set as a multiple of three (e.g., $D = 12, 27, 90$). Algorithm 6 shows the calculation method of the objective function value of the D-dimensional order-3 deceptive problem. Clearly, the optimal solution of this problem is $\boldsymbol{x}^* = (1, ..., 1)^{\mathrm{T}}$.

2.5.1 Exercise 5: Implement $(\mu + \lambda)$-GA

The goal of this exercise is to implement $(\mu + \lambda)$-GA, which is a generalized version of $(\mu + 1)$-GA. While only one child is generated per one iteration in $(\mu + 1)$-GA, λ children are generated simultaneously in $(\mu + \lambda)$-GA. Although there have been a number of variants of $(\mu + \lambda)$-GA, a $(\mu + \lambda)$-GA with truncation selection like CHC [1] is considered in this exercise. Algorithm 7 shows the details of $(\mu + \lambda)$-GA to be implemented.

Algorithm 7: $(\mu + \lambda)$-GA

```
   //  Initialization phase
1  t ← 1;
2  Initialize the population P = {x^1, ..., x^μ} randomly (Algorithm 2);
3  Evaluate each individual x^i (i ∈ {1, ..., μ}) in P by equation (1);
4  while The termination criteria are not met do
5      Children Q ← ∅;
6      for i ∈ {1, ..., λ} do
           //  Step 1.  Mating selection
7          Randomly select two parent individuals x^a and x^b from P so that a ≠ b;
           //  Step 2.  Crossover operator
8          Generate a child u by applying uniform crossover (Algorithm 3) to x^a and x^b;
           //  Step 3.  Mutation operator
9          Apply bit-flip mutation (Algorithm 4) with the mutation probability p_m = 1/D to
           u;
           //  Step 4.  Evaluate the child u
10         Evaluate the child u by equation (1);
11         n ← n + 1;
12         Q ← Q ∪ {u};
           //  Step 5.  Update the best-so-far solution x^bsf
13         if f(u) > f(x^bsf) then
14             x^bsf ← u;
       //  Step 6.  Environmental selection (R is a union of the population and
           children)
15     R ← P ∪ Q, where R = {x^1, ..., x^μ, u^1, ..., u^λ};
16     Sort individuals in R according to their objective function values;
17     P ← the first half of individuals in R;
18     t ← t + 1;
19 Show the best-so-far solution x^bsf and its objective function value f(x^bsf) to an user (you);
```

3 GA for permutation optimization problems

In permutation optimization problems, the solution is defined as a permutation such as $\boldsymbol{x} = (3, 1, 4, 2)^{\mathrm{T}}$. Here, the TSP, which is the most popular permutation optimization problem, is considered.

In addition to the TSP, there are a number of problems such as Flow Shop Scheduling Problems (FSSPs), Quadratic Assignment Problems (QAPs), and Vehicle Routing Problems (VRPs). QAPs are easy to understand, and a GA for them is also easy to implement. If you are interested in the GA for QAPs, you can try it after finishing all exercises.

3.1 Traveling Sales-person Problems (TSPs)

The Traveling Sales-person Problem (TSP) is the problem of finding the shortest tour that visits all cities. The TSP is not described anymore here because it is very famous. You can find good examples of TSPs anywhere on the internet.

3.2 Exercise 6: Implement GA for solving TSPs

The goal of this section is to implement a GA for TSPs. Both $(\mu + 1)$-GA and $(\mu + \lambda)$-GA are fine. The algorithm 8, 9, and 10 show the calculation method of the tour length, the procedure of Order Crossover (OX) and inversion mutation, respectively. Although there have been a number of crossover methods for permutation problems, OX is easy to understand and works well on small-scale TSPs. Figure 2 in [4] shows a good example of the procedure of OX. You should use Fisher-Yates's shuffle method[1] or its variants for randomly generating the initial solution $\boldsymbol{x}$.

Which TSP instances should be used? TSP instances provided by TSPLIB[2] are the most commonly used benchmark problems. However, data of TSP instances cannot currently be downloaded from the TSPLIB website (Aug 20th, 2017). Therefore, instead of TSPLIB, you can use data of national TSPs[3]. Here, the following two TSP instances should be considered as benchmark problems: the Western Sahara's 29 cities problem (wi29) and the Djibouti's 38 cities problem (dj38). Their data can be downloaded from `http://www.math.uwaterloo.ca/tsp/world/wi29.tsp` and `http://www.math.uwaterloo.ca/tsp/world/dj38.tsp`, respectively. Note that it is impossible for a GA without well-designed strategies to solve large-scale TSPs (e.g., the China's 71,009 cities problem).

There are two ways to store TSP data to your program. The first way is to save TSP instance data into a file and read it from the program. The second way is to directly save TSP instance data to you program as follows:

```
If (target_TSP_instance == ‘‘wi29’’) {
  tsp_data_x = {20833.3333, 20900, ..., 27462.5};
  tsp_data_y = {17100, 17066.6667, ..., 12992.2222};
}
Else if (target_TSP_instance == ‘‘dj38’’) {
  tsp_data_x = {11003.6111, 11108.6111, ..., 12645};
  tsp_data_y = {42102.5, 42373.8889, ..., 42973.3333};
}
```

[1] `https://en.wikipedia.org/wiki/Fisher-Yates_shuffle`

[2] `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

[3] `http://www.math.uwaterloo.ca/tsp/world/countries.html`

Algorithm 8: The calculation method of the tour length (the objective function value $f(\boldsymbol{x})$) of the solution $\boldsymbol{x}$. The D is the number of given cities. The $\boldsymbol{d}$ is a $D \times D$ matrix whose each element represents the Euclidean distance between two cities. For example, $d_{3,8}$ is the distance between cities 3 and 8.

$L \leftarrow 0$;
for $j = 1$ **to** $D - 1$ **do**
    $L \leftarrow L + d_{x_j, x_{j+1}}$;
$L \leftarrow L + d_{x_D, x_1}$;

Algorithm 9: The procedure of order crossover (OX).

Randomly select two crossover points c_1 and c_2 from $\{1, ..., D\}$ so that $c_1 < c_2$;
$\boldsymbol{u} \leftarrow \boldsymbol{x}^a$;
for $j = c_1$ **to** c_2 **do**
    $h \leftarrow 1$;
    while $x_j^b \neq u_h$ **do**
        $h \leftarrow h + 1$;
    $l \leftarrow h + 1$;
    while $l \neq c_2$ **do**
        if $h = D$ **then**
            $h \leftarrow 1$;
        if $l = D$ **then**
            $l \leftarrow 1$;
        $u_h \leftarrow u_l$;
        $h \leftarrow h + 1$;
        $l \leftarrow l + 1$;
for $j = c_1$ **to** c_2 **do**
    $u_j \leftarrow x_j^b$;

Algorithm 10: The procedure of inversion mutation.

Randomly select two mutation points m_1 and m_2 from $\{1, ..., D\}$ so that $m_1 < m_2$;
$h \leftarrow (m_2 - m_1)/2$;
$m_1 \leftarrow m_1 + 1$;
for $k = 1$ **to** h **do**
    $l \leftarrow u_{m_1+k}$;
    $u_{m_1+k} \leftarrow u_{m_2-k}$;
    $u_{m_2-k} \leftarrow l$;

4 GA for Continuous Optimization

4.1 Continuous Optimization

Continuous optimization, which frequently appears in the field of engineering optimization, is the problem of finding a D-dimensional solution $\boldsymbol{x} = (x_1, ..., x_D)^{\mathrm{T}} \in \mathbb{S} \subseteq \mathbb{R}^D$ that minimizes an objective function $f : \mathbb{R}^D \to \mathbb{R}$. $\mathbb{S} = \Pi_{j=1}^{D}[x_j^{\min}, x_j^{\max}]$ is the bound-constrained search space where $x_j^{\min} \leq x_j \leq x_j^{\max}$ for each index $j \in \{1, ..., D\}$. While decision variables are discrete in combinatorial optimization (including binary and permutation optimization problems), those are real-coded values in continuous optimization.

4.2 Exercise 7: Implement a GA for continuous optimization

The goal of this exercise is to implement a GA for solving continuous optimization problems. Both $(\mu+1)$-GA and $(\mu+\lambda)$-GA are fine. The following three test functions (the Sphere, Rastrigin, and Rosenbrock functions) should be used for the performance evaluation of your GA:

• The Sphere function:

$$f_{\mathrm{Sphere}}(\boldsymbol{x}) = \sum_{j=1}^{D} x_j^2 \tag{3}$$

◦ $\boldsymbol{S} = [-100, 100]^D$, $\boldsymbol{x}^* = (0, ..., 0)^{\mathrm{T}}$ and $f(\boldsymbol{x}^*) = 0$, Function properties: Unimodal, separable

• The Rastrigin function:

$$f_{\mathrm{Rastrigin}}(\boldsymbol{x}) = \sum_{j=1}^{D} \big(x_j^2 - 10\mathrm{cos}2\pi(x_j) + 10\big) \tag{4}$$

◦ $\boldsymbol{S} = [-5.12, 5.12]^D$, $\boldsymbol{x}^* = (0, ..., 0)^{\mathrm{T}}$ and $f(\boldsymbol{x}^*) = 0$, Function properties: Strong multimodal, separable

• The Rosenbrock function:

$$f_{\mathrm{Rosenbrock}}(\boldsymbol{x}) = \sum_{j=1}^{D-1} \big(100(x_{j+1} - x_j^2)^2 + (x_j - 1)^2\big) \tag{5}$$

◦ $\boldsymbol{S} = [-30, 30]^D$, $\boldsymbol{x}^* = (1, ..., 1)^{\mathrm{T}}$ and $f(\boldsymbol{x}^*) = 0$, Function properties: Weak multimodal, nonseparable

The algorithm 11 shows how to randomly generate individuals of the initial population. Although a number of crossover methods have been proposed for continuous optimization, Blend Crossover α (BLX-α) [2][4] is easy to implement and performs well. Note that a mutation operator is not applied to a child in this exercise.

For two parent individuals $\boldsymbol{x}^a$ and $\boldsymbol{x}^b$, BLX-α generates each element of a child u_j ($j \in \{1, ..., D\}$) in the α times extended range in $[x_j^a, x_j^b]$ as follows:

$$u_j = \mathrm{rand}[A_j, B_j] \tag{6}$$

$$A_j = \min(x_j^a, x_j^b) - \alpha\,|x_j^a - x_j^b| \tag{7}$$

$$B_j = \max(x_j^a, x_j^b) + \alpha\,|x_j^a - x_j^b| \tag{8}$$

[4]While I have seen a large number of articles that cite [2], I have never seen the original paper. It could not be found anywhere.

Algorithm 11: The initialization method of the population $\boldsymbol{P} = \{\boldsymbol{x}^1, ..., \boldsymbol{x}^\mu\}$, $\boldsymbol{x} = (x_1, ..., x_D)^{\mathrm{T}}$, where $x^{\max}$ and $x^{\min}$ are the maximum and minimum values of the search space.

```
for i ∈ {1, ..., μ} do
    for j ∈ {1, ..., D} do
        x_j^i = (x^max − x^min)rand[0, 1] + x^min;
```

where $\alpha > 0$ is the expansion rate, and its recommended setting is 0.5. The functions $\min(\cdot, \cdot)$ and $\max(\cdot, \cdot)$ return the smallest and biggest values of given two values, respectively. For example, $\min(1.2, 3.4) = 1.2$ and $\max(1.2, 3.4) = 3.4$. Good examples of BLX-α can be found in the internet.

It is possible that a child generated by equation (6) violates the bound constraint $\mathbb{S} = \Pi_{j=1}^{D}[x_j^{\min}, x_j^{\max}]$. In such case, the following repairing method should be applied:

$$u_j = \begin{cases} x_j^{\min} & \text{if } u_j < x_j^{\min} \\ x_j^{\max} & \text{if } u_j > x_j^{\max} \\ u_j & \text{otherwise} \end{cases} \tag{9}$$

For example, if $u_5 = 123.45$, it should be 100.

If the GA which you implemented can successfully find the optimal solution[5] of the three test functions with $D = 5$, this exercise is finished.

References

[1] L. J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In *FOGA*, pages 265–283, 1990.

[2] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In *FOGA*, pages 187–202, 1992.

[3] M. Li, C. O'Riordan, and S. Hill. An analysis of multi-chromosome GAs in deceptive problems. In *GECCO*, pages 1021–1028, 2011.

[4] T. Starkweather, S. McDaniel, K. E. Mathias, L. D. Whitley, and C. Whitley. A Comparison of Genetic Sequencing Operators. In *ICGA*, pages 69–76, 1991.

[5] If $|f(\boldsymbol{x}) - f(\boldsymbol{x}^*)| \leq 10^{-8}$, the solution $\boldsymbol{x}$ can be treated as the optimal solution.