The answer sheet for evolutionary computation

Wenxiang Chen
PB07210425

August 18, 2010

1. In an evolutionary algorithm where all chromosomes are represented by binary strings of length n, k-point crossover ($1 \leq k \leq n-1$) and uniform crossover are often used.

   (a) Given two chromosomes, describe how a 3-point crossover works (Assume that the chromosome length is greater than 3). You can use a pseudo-code procedure or an example to illustrate it.
   Solution:

   - The pseudo-code on how 3-point crossover works[3] is listed in Algorithm 2:

Algorithm 1: How 3-point crossover operator works

Input: two given parents $A^{(t)}$ and $B^{(t)}$
Output: two offspring $C^{(t)}$ and $D^{(t)}$ after 3-point crossover

randomly choose 3 crossover cp_1, cp_2, cp_3 from set $\{1, \cdots, n-1\}$;
for $i \longleftarrow 1$ **to** cp_1 **do**
    $C_i^{(t+1)} \longleftarrow A_i^{(t)}$;
    $D_i^{(t+1)} \longleftarrow B_i^{(t)}$;
$flag \longleftarrow 0$;
for $j \longleftarrow 2$ **to** 3 **do**
    for $i \longleftarrow cp_{j-1} + 1$ **to** cp_j **do**
        $C_i^{(t+1)} \longleftarrow flag \times A_i^{(t)} + (1 - flag) \times B_i^{(t)}$;
        $D_i^{(t+1)} \longleftarrow (1 - flag) \times A_i^{(t)} + flag \times B_i^{(t)}$;
        $flag \longleftarrow 1 - flag$;
for $i \longleftarrow cp_3 + 1$ **to** n **do**
    $C_i^{(t+1)} \longleftarrow flag \times A_i^{(t)} + (1 - flag) \times B_i^{(t)}$;
    $D_i^{(t+1)} \longleftarrow (1 - flag) \times A_i^{(t)} + flag \times B_i^{(t)}$;

   - An example on how a 3-point crossover works is shown in Figure 2.

   (b) Given two chromosomes, decribe how the uniform crossover works. You can use a pseudo-code or an example to illustrate it.
   Solution:

   - The pseudo-code on how uniform crossover works [4] is listed in Algorithm 2:
   - An example on how a uniform crossover works is shown in Figure 3.

   (c) Is the $(n-1)$-point crossover the same as the uniform crossover? If not, explain the differences.
   Solution: $(n-1)$-point crossover and uniform crossover are different for several reasons:

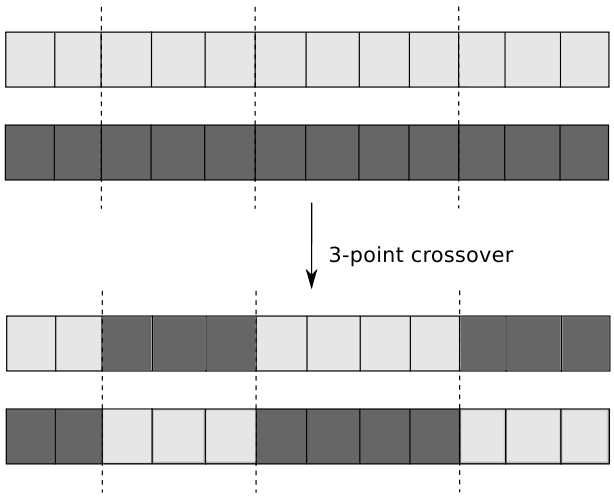

Figure 1: 3-Point Crossover

Figure 2: How 3-point crossover operator works

Algorithm 2: How uniform crossover operator works

Input: two given parents $A^{(t)}$ and $B^{(t)}$
Output: two offspring $C^{(t)}$ and $D^{(t)}$ after 3-point crossover

for $i \longleftarrow 1$ **to** n **do**
    choose a uniform random real-value number u from interval $[0, 1]$;
    if $u \leq p_s$ **then**
        // p_s is the probability of swapping, often set to 0.5
        $C_i^{(t+1)} \longleftarrow A_i^{(t)}$;
        $D_i^{(t+1)} \longleftarrow B_i^{(t)}$;
    else
        $C_i^{(t+1)} \longleftarrow B_i^{(t)}$;
        $D_i^{(t+1)} \longleftarrow A_i^{(t)}$;

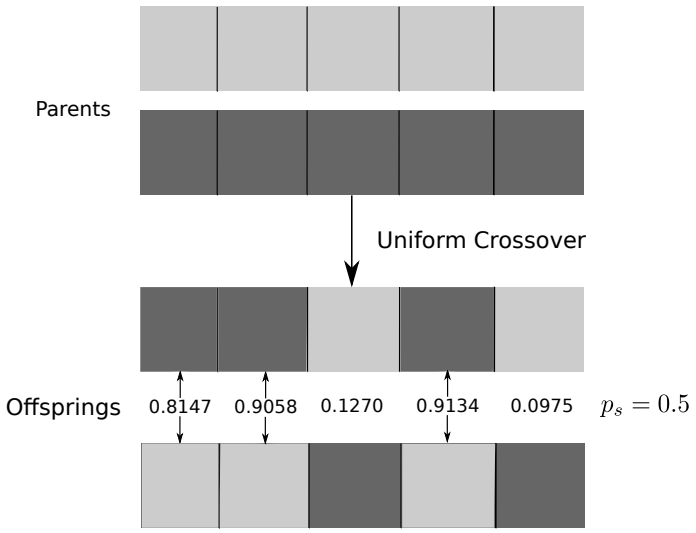

Figure 3: Uniform Crossover

- On one hand, $(n-1)$-point crossover is a deterministic operator. The offspring are generated by selecting from each other genes from the two parent. For instance, for a certain offspring, if its consecutively previous gene is from parent 1, then its current gene should be from parent 2.
- On the other hand, 'uniform' from uniform crossover means every gene in a certain offspring has the same probability to inherit gene from a specified parent. In order to produce offspring, uniform crossover operator need a series of uniform distributed random numbers, which is unknown beforehand.

2. Answer the following questions:

   (a) Given N chromosomes in a population and their fitness values, $f_1, f_2, \cdots, f_N$, describe the probability of selecting the i-th chromosome $(1 \leq i \leq N)$ using the roulette-wheel selection.
   Solution: Suppose the probability of selecting the i-th chromosome $(1 \leq i \leq N)$ is notated as p_i, it can be calculated as shown in Equation 1

$$p_i = \frac{f_i}{\Sigma_{j=1}^{N} f_j} \tag{1}$$

   where f_i $(1 \leq i \leq N)$ is the fitness of i-th chromosome.

   (b) Roulette-wheel selection may lead to premature convergence of an evolutionary algorithm due to the "super-individual" problem. Explain why this problem is, using the definition of roulette-wheel selection and an example.
   Solution: For a maximization problem, suppose the fitness is defined as Equation 2.

$$\{f(x)|f(x) = x^4, x \in [-4, 4], x\, is\, an\, integer\} \tag{2}$$

   The population $p\vec{o}p$ is 0, -1, 1, -3. The fitness for each individual $f(p\vec{o}p)$ is 0, 1, 1, 81. According to 1, the probability of selecting the i-th chromosome $\vec{p}$ is 0, 0.0120, 0.0120, 0.9759. Obviously, the 4-th individual dominate the population, and all the other individual have little chance to be selected in roulette-wheel selection. Consequently, the roulette-wheel selection can lead to premature convergence in such case.

(c) In tournament selection, the size of tournament is an important parameter. If we want to increase the selection pressure, i.e., making good chromosomes more likely to be selected as parents, should we increase or decrease the tournament size?
Solution: We should increase the tournament size for increasing the selection pressure. In tournament selection [2], a number (i.e., tournament size) of individuals is chosen randomly from the population and the best individual from this group is selected as parent. Therefore, the larger the tournament size is, the more 'global' will the winner from a certain tournament be, which makes the pressure of selection stronger.

(d) What happens when the tournament size is the same as the population size?
Solution: When the tournament size is the same as the population size, tournament selection will always select the best individual as parent.

3. Given an evolutionary optimization algorithm where all chromosomes are represented by real-valued vectors.

(a) Design an appropriate multi-parent (i.e., more than two parents) crossover operator.
Solution: Here I would like to introduce a straightforward approach for multi-parent crossover procedure.

i. m individuals $\vec{x}_j$ are uniform randomly selected from the population, $j \in 1, 2, \cdots, m$.

ii. m uniform distributed random real number α_j, $j \in 1, 2, \cdots, m$ are subject to Equation 3.

$$\sum_{j}^{m} \alpha_j = 1 \tag{3}$$

where ε is a positive parameter,$-\varepsilon \leq \alpha_j \leq 1 + \varepsilon$ and $j \in 1, 2, \cdots, m$.

iii. the children $\vec{x'}$ is produced as in Equation 4

$$\vec{x'} = \sum_{j}^{m} \alpha_j \vec{x}_j \tag{4}$$

Explanation: The basic idea of this multi-parent crossover operator is linear combination of several parents real-value vector. The advantage of this operator is that it would provide broader choice for combination. It becomes possible that merits from different parents will mix and cooperate together. Moreover, the parameter ε in Equation 3 enables the ability to search area outside where parents reside.

(b) Design an appropriate self-adaptive mutation operator.
Solution: Motivated by Differential Evolution[5], here I introduce an appropriate self-adaptive mutation operator. For every candidate solution $\vec{x_{i,G}}$, $i = 1, 2, \cdots, NP$, the mutation is generated according to Equation 5

$$\vec{v}_{i,G+1} = \vec{x}_{r_1,G} + F_i \times (\vec{x}_{r_2,G} - \vec{x}_{r_3,G}) \tag{5}$$

where F_i is defined in Equation 6

$$F_i = randn(0, mean(\vec{F}_{i,success})^2) \tag{6}$$

we record all the corresponding F_i that produce a success mutation in the vector $\vec{F}_{i,success}$. F_i is randomly sampled from a Gaussian distribution whose mean is 0 and standard deviation is $mean(\vec{F}_{i,success})$
Explanation: Uniform randomly selected indices $r_1, r_2, r_3 \in \{1, 2, \cdots, NP\}$ are integer, mutually different and $F_i > 0$, is the scale factor of differential variation. Besides, r_1, r_2, r_3 should be different from i. F_i is set initially to 0.5, and keeps self-adaptive during optimization. Since current F_i is averaged over all those previous F_i related with successful mutation, it is able to self-adapt to different stage of evolution.

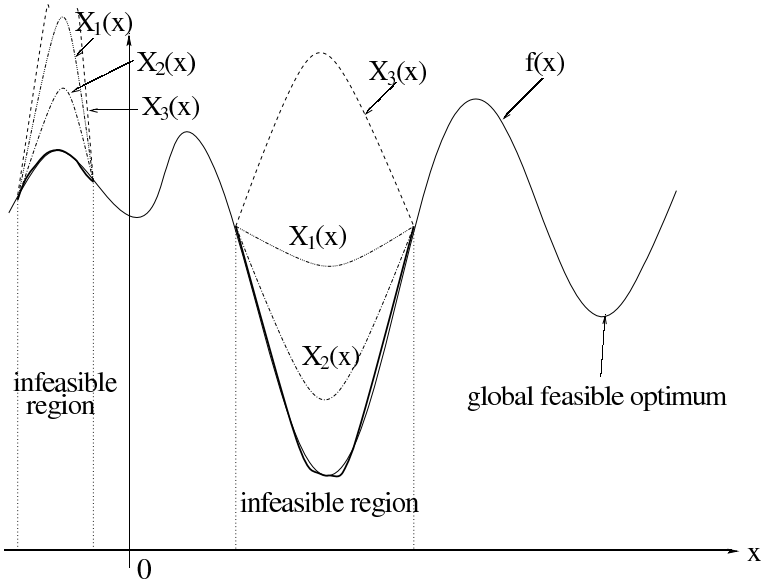

Figure 4: The example for penalty function

4. Penalty functions have often been used in constrained optimization to convert a constrained problem into an unconstrained one.

   (a) Discuss the major advantages and potential problems of the penalty function approach to constrained optimization.
   Solution:

   Major Advantage Penalty function approach allow the algorithm to explore infeasible search space. The algorithm would be able gather the global information from the entire search space. Consequently, it would be beneficial for seeking for those optimum which are located near the boundaries between feasible and infeasible search space.

   Potential Problem It is hard to determine the penalty coefficients without domain knowledge, and an inappropriate setting for penalty coefficient will significantly affect the quality of solution.

   (b) Explain why and how an inappropriate penalty function can make finding the constrained global optimum difficult. You may use an example to illustrate the reason.
   Solution: As shown in Figure 4, suppose that it is a minimization problem.

   - if the penalty coefficient is too large, as the $X_3(x)$ shows, the penalty part would have a remarkable effect for fitness. As a result, the penalty part would make the fitness landscape even more complex.
   - on the other hand, if the penalty coefficient is too small, as $X_2(x)$ shows, the algorithm is still possible stop with a solution that is located in infeasible search space.

   (c) When a penalty function is used, can we guarantee that a feasible solution will be found? Explain your answer.
   Solution: As it is shown in Figure 4, if the coefficient is set inappropriately, say too small, it is still possible that the best solution found by algorithm is an infeasible solution.

5. Given a binary classification problem with a training data set D, design an evolutionary learning algorithm that evolves a set of rules that best solve the problem. The following should be included in your answer.

(a) Encoding of a rule set in a chromosome

(b) One or more appropriate crossover operators

(c) One or more appropriate mutation operators

(d) An appropriate fitness definition
Solution: As it is known to us , the Pitt approach tend to be better suited at batch-mode learning [1] and the training set D is available before learning is initiated. Here we choose Pitt approach as our basic principle for evolving. To make our algorithm more general, and more suitable for real-world application, we assume that the attributes of given D are real numbers.

Encoding In our encoding method, each chromosome represents a set of rules. Every rule contains a set of A attributes and a class label. Each attribute in the rule has two real numbers which represents the minimum and maximum of the valid range for the corresponding attribute. We use a fixed-length chromosome for GA, where the length of a chromosome is $n \times (2 \times A + 1)$.

Crossover We employ the simple k-point crossover operator, where k is integer, which is uniform randomly generated from $[1, n-1]$.

Mutation We employ two different kinds mutation operator in our algorithm:

- Creep mutation: Creep mutation is applied to each attribute denoted as the creep rate. Attribute values are incremented or decremented by the creep fraction, which is a fraction of the valid range for the attribute value. It is equally likely that the attribute value will be incremented or decremented. Please note that creep mutation makes it possible for the range specified in the chromosome for the attribute to change from a normal range to a invalid range or vice versa.
- Simple random mutation: Simple random mutation replaces the uniform randomly chosen attribute range values with random values from the appropriate valid range or class label with opposite class (since it is binary classification problem).

Fitness Definition Since every chromosome in our representation approach comprises an entire rule-based system, the fitness function must evaluate the collective effectiveness of the rule set. Here in our algorithm, the fitness is defined by the accuracy of the chromosome's rule set, which is a real value ranging from 0 to 1.

Conflict Resolution As there is more one rule, it is possible for multiple rules to match a particular given instance yet predict contradictory classification. Thus, we need the mechanism to resolute conflict in such cases. We apply a weighted voting strategy in our algorithm:

- A history of the amount of correct and incorrect matches is maintained for each rule.
- Each matching rule votes for its class by adding the amount of correct and incorrect matches to the corresponding variables for its class.
- The first objective is to select the class with the least number of incorrect matches by rules predicting that class. If there is a tie between two classes under this measurement, the second objective is to select the class with the greatest number of correct matches. If there is still a tie, we pick one of those rule sets uniform randomly.

References

[1] Arthur L. Corcoran and Sandip Sen. Using real-valued genetic algorithms to evolve rule sets for classification. In *In IEEE-CEC*, pages 120–124, 1994.

[2] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[3] Tatsuya Nomura. An analysis on crossovers for real number chromosomes in an infinite population size. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 936–941, 1997.

[4] Villiam M. Spears and Kenneth A. De Jong. On the virtues of parameterized uniform crossover. In *In Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.

[5] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.