3.2 Genetic Algorithm

Genetic Algorithm, GA, Simple Genetic Algorithm, SGA, Canonical Genetic Algorithm, CGA.

3.2.1 Taxonomy

The Genetic Algorithm is an Adaptive Strategy and a Global Optimization technique. It is an Evolutionary Algorithm and belongs to the broader study of Evolutionary Computation. The Genetic Algorithm is a sibling of other Evolutionary Algorithms such as Genetic Programming (Section 3.3), Evolution Strategies (Section 3.4), Evolutionary Programming (Section 3.6), and Learning Classifier Systems (Section 3.9). The Genetic Algorithm is a parent of a large number of variant techniques and sub-fields too numerous to list.

3.2.2 Inspiration

The Genetic Algorithm is inspired by population genetics (including heredity and gene frequencies), and evolution at the population level, as well as the Mendelian understanding of the structure (such as chromosomes, genes, alleles) and mechanisms (such as recombination and mutation). This is the so-called new or modern synthesis of evolutionary biology.

3.2.3 Metaphor

Individuals of a population contribute their genetic material (called the genotype) proportional to their suitability of their expressed genome (called their phenotype) to their environment, in the form of offspring. The next generation is created through a process of mating that involves recombination of two individuals genomes in the population with the introduction of random copying errors (called mutation). This iterative process may result in an improved adaptive-fit between the phenotypes of individuals in a population and the environment.

3.2.4 Strategy

The objective of the Genetic Algorithm is to maximize the payoff of candidate solutions in the population against a cost function from the problem domain. The strategy for the Genetic Algorithm is to repeatedly employ surrogates for the recombination and mutation genetic mechanisms on the population of candidate solutions, where the cost function (also known as objective or fitness function) applied to a decoded representation of a candidate governs the probabilistic contributions a given candidate solution can make to the subsequent generation of candidate solutions.

3.2.5 Procedure

Algorithm 3.2.1 provides a pseudocode listing of the Genetic Algorithm for minimizing a cost function.

Algorithm 3.2.1: Pseudocode for the Genetic Algorithm.

Input: $Population_{size}$, $Problem_{size}$, $P_{crossover}$, $P_{mutation}$
Output: S_{best}

```
Population ← InitializePopulation(Population_size, Problem_size);
EvaluatePopulation(Population);
S_best ← GetBestSolution(Population);
while ¬StopCondition() do
    Parents ← SelectParents(Population, Population_size);
    Children ← ∅;
    foreach Parent_1, Parent_2 ∈ Parents do
        Child_1, Child_2 ← Crossover(Parent_1, Parent_2, P_crossover);
        Children ← Mutate(Child_1, P_mutation);
        Children ← Mutate(Child_2, P_mutation);
    end
    EvaluatePopulation(Children);
    S_best ← GetBestSolution(Children);
    Population ← Replace(Population, Children);
end
return S_best;
```

3.2.6 Heuristics

- Binary strings (referred to as 'bitstrings') are the classical representation as they can be decoded to almost any desired representation. Real-valued and integer variables can be decoded using the binary coded decimal method, one's or two's complement methods, or the gray code method, the latter of which is generally preferred.
- Problem specific representations and customized genetic operators should be adopted, incorporating as much prior information about the problem domain as possible.
- The size of the population must be large enough to provide sufficient coverage of the domain and mixing of the useful sub-components of the solution [7].
- The Genetic Algorithm is classically configured with a high probability of recombination (such as 95%-99% of the selected population) and

a low probability of mutation (such as $\frac{1}{L}$ where L is the number of components in a solution) [1, 18].

- The fitness-proportionate selection of candidate solutions to contribute to the next generation should be neither too greedy (to avoid the takeover of fitter candidate solutions) nor too random.

3.2.7 Code Listing

Listing 3.1 provides an example of the Genetic Algorithm implemented in the Ruby Programming Language. The demonstration problem is a maximizing binary optimization problem called OneMax that seeks a binary string of unity (all '1' bits). The objective function provides only an indication of the number of correct bits in a candidate string, not the positions of the correct bits.

The Genetic Algorithm is implemented with a conservative configuration including binary tournament selection for the selection operator, one-point crossover for the recombination operator, and point mutations for the mutation operator.

```
def onemax(bitstring)
  sum = 0
  bitstring.size.times {|i| sum+=1 if bitstring[i].chr=='1'}
  return sum
end

def random_bitstring(num_bits)
  return (0...num_bits).inject(""){|s,i| s<<((rand<0.5) ? "1" : "0")}
end

def binary_tournament(pop)
  i, j = rand(pop.size), rand(pop.size)
  j = rand(pop.size) while j==i
  return (pop[i][:fitness] > pop[j][:fitness]) ? pop[i] : pop[j]
end

def point_mutation(bitstring, rate=1.0/bitstring.size)
  child = ""
   bitstring.size.times do |i|
     bit = bitstring[i].chr
     child << ((rand()<rate) ? ((bit=='1') ? "0" : "1") : bit)
  end
  return child
end

def crossover(parent1, parent2, rate)
  return ""+parent1 if rand()>=rate
  point = 1 + rand(parent1.size-2)
  return parent1[0...point]+parent2[point...(parent1.size)]
end

def reproduce(selected, pop_size, p_cross, p_mutation)
```

```
  children = []
  selected.each_with_index do |p1, i|
    p2 = (i.modulo(2)==0) ? selected[i+1] : selected[i-1]
    p2 = selected[0] if i == selected.size-1
    child = {}
    child[:bitstring] = crossover(p1[:bitstring], p2[:bitstring], p_cross)
    child[:bitstring] = point_mutation(child[:bitstring], p_mutation)
    children << child
    break if children.size >= pop_size
  end
  return children
end

def search(max_gens, num_bits, pop_size, p_crossover, p_mutation)
  population = Array.new(pop_size) do |i|
    {:bitstring=>random_bitstring(num_bits)}
  end
  population.each{|c| c[:fitness] = onemax(c[:bitstring])}
  best = population.sort{|x,y| y[:fitness] <=> x[:fitness]}.first
  max_gens.times do |gen|
    selected = Array.new(pop_size){|i| binary_tournament(population)}
    children = reproduce(selected, pop_size, p_crossover, p_mutation)
    children.each{|c| c[:fitness] = onemax(c[:bitstring])}
    children.sort!{|x,y| y[:fitness] <=> x[:fitness]}
    best = children.first if children.first[:fitness] >= best[:fitness]
    population = children
    puts " > gen #{gen}, best: #{best[:fitness]}, #{best[:bitstring]}"
    break if best[:fitness] == num_bits
  end
  return best
end

if __FILE__ == $0
  # problem configuration
  num_bits = 64
  # algorithm configuration
  max_gens = 100
  pop_size = 100
  p_crossover = 0.98
  p_mutation = 1.0/num_bits
  # execute the algorithm
  best = search(max_gens, num_bits, pop_size, p_crossover, p_mutation)
  puts "done! Solution: f=#{best[:fitness]}, s=#{best[:bitstring]}"
end
```

Listing 3.1: Genetic Algorithm in Ruby

3.2.8 References

Primary Sources

Holland is the grandfather of the field that became Genetic Algorithms. Holland investigated adaptive systems in the late 1960s proposing an adaptive system formalism and adaptive strategies referred to as 'adaptive plans'

[8–10]. Holland's theoretical framework was investigated and elaborated by his Ph.D. students at the University of Michigan. Rosenberg investigated a chemical and molecular model of a biological inspired adaptive plan [19]. Bagley investigated meta-environments and a genetic adaptive plan referred to as a genetic algorithm applied to a simple game called hexapawn [2]. Cavicchio further elaborated the genetic adaptive plan by proposing numerous variations, referring to some as 'reproductive plans' [15].

Other important contributions were made by Frantz who investigated what were referred to as genetic algorithms for search [3], and Hollstien who investigated genetic plans for adaptive control and function optimization [12]. De Jong performed a seminal investigation of the genetic adaptive model (genetic plans) applied to continuous function optimization and his suite of test problems adopted are still commonly used [13]. Holland wrote the the seminal book on his research focusing on the proposed adaptive systems formalism, the reproductive and genetic adaptive plans, and provided a theoretical framework for the mechanisms used and explanation for the capabilities of what would become genetic algorithms [11].

Learn More

The field of genetic algorithms is very large, resulting in large numbers of variations on the canonical technique. Goldberg provides a classical overview of the field in a review article [5], as does Mitchell [16]. Whitley describes a classical tutorial for the Genetic Algorithm covering both practical and theoretical concerns [20].

The algorithm is highly-modular and a sub-field exists to study each sub-process, specifically: selection, recombination, mutation, and representation. The Genetic Algorithm is most commonly used as an optimization technique, although it should also be considered a general adaptive strategy [14]. The schema theorem is a classical explanation for the power of the Genetic Algorithm proposed by Holland [11], and investigated by Goldberg under the name of the building block hypothesis [4].

The classical book on genetic algorithms as an optimization and machine learning technique was written by Goldberg and provides an in-depth review and practical study of the approach [4]. Mitchell provides a contemporary reference text introducing the technique and the field [17]. Finally, Goldberg provides a modern study of the field, the lessons learned, and reviews the broader toolset of optimization algorithms that the field has produced [6].

3.2.9 Bibliography

[1] T. Bäck. Optimal mutation rates in genetic search. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 2–9, 1993.

[2] J. D. Bagley. *The behavior of adaptive systems which employ genetic and correlation algorithms*. PhD thesis, University of Michigan, 1967.

[3] D. R. Frantz. *Non-linearities in genetic adaptive search*. PhD thesis, University of Michigan, 1972.

[4] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[5] D. E. Goldberg. Genetic and evolutionary algorithms come of age. *Communications of the ACM*, 37(3):113–119, 1994.

[6] D. E. Goldberg. *The design of innovation: Lessons from and for competent genetic algorithms*. Springer, 2002.

[7] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362, 1992.

[8] J. H. Holland. Information processing in adaptive systems. In *Processing of Information in the Nervous System*, pages 330–338, 1962.

[9] J. H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314, 1962.

[10] J. H. Holland. Adaptive plans optimal for payoff-only environments. In *Proceedings of the Second Hawaii Conference on Systems Sciences*, 1969.

[11] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[12] R. B. Hollstien. *Artificial genetic adaptation in computer control systems*. PhD thesis, The University of Michigan, 1971.

[13] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan Ann Arbor, MI, USA, 1975.

[14] K. A. De Jong. Genetic algorithms are NOT function optimizers. In *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, pages 5–17. Morgan Kaufmann, 1992.

[15] D. J. Cavicchio Jr. *Adaptive Search Using Simulated Evolution*. PhD thesis, The University of Michigan, 1970.

[16] M. Mitchell. Genetic algorithms: An overview. *Complexity*, 1(1):31–39, 1995.

[17] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[18] H. Mühlenbein. How genetic algorithms really work: I. mutation and hillclimbing. In *Parallel Problem Solving from Nature 2*, pages 15–26, 1992.

[19] R. Rosenberg. *Simulation of genetic populations with biochemical properties*. PhD thesis, University of Michigan, 1967.

[20] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.