

CS341 SYSTEMS BIOLOGY I

LECTURE 18

GENETIC ALGORITHMS FOR PARAMETER-FITTING

Recall that we need a cost function whose input is a set of parameters and whose output is a single number ≥ 0 , where larger numbers indicate worse costs. We then search for a set of parameter values that cause the cost function to be as small as possible. In other words, this is a numerical minimization problem. We also call it optimization (we are looking for an optimal set of parameters).

Today, we use a bio-inspired algorithm - one that mimics the process of evolution. We begin with an entire population of initial guesses (each parameter set is an “individual”) and then use the process of mating with selection and mutation to generate new parameter sets (individuals). The process of selection guides the population to more fit (less costly) individuals and the process of mutation allows the algorithm to escape local minima.

There are many, many variants of genetic algorithms. We will be starting with the following algorithm, then quickly moving on to one that is slightly more complex. The basic algorithm uses selection to create a breeding pool P from each generation. For each individual in a new generation, 2 parents are chosen from that breeding pool, elements of the parents are used to create the individual (cross-over), and then those elements are mutated.

- Create an initial population of λ individuals G^0 and generate their costs.
- Sort the individuals in G^0 by cost (in preparation for selection)
- For each generation g
 - **Select** the breeding pool P of μ individuals from the previous generation. There are different strategies for selection and they depend on the cost. Individuals are more likely to join P if they are low cost.
 - For i in $\text{range}(\lambda)$
 - * Randomly choose two parents from P .
 - * Use **cross-over** to generate a child G_i^g . This means we take elements of each parent parameter set to make the child set.
 - * **Mutate** the values in G_i^g . This means we make small changes to each element of the child parameter set.
 - * Compute the child's cost.
 - Sort the individuals in G_i^g by cost (in preparation for selection)

This works well, but has the drawback that if we find super low-cost individuals along the way, there is no guarantee they will appear in the final generation. If we have a great solution, why throw it away? One way to address that is simply to make it so “elite” individuals to just join the next generation. Here is the algorithm with elitism added in:

- Create an initial population of λ individuals G^0 and generate their costs.
- Sort the individuals in G^0 by cost (in preparation for selection)
- For each generation g
 - **Elitism:** Take the *eliteCount* best children from the previous generation G^{g-1} and put them into this generation G^g (G_0^g through $G_{\text{eliteCount}}^g$)
 - **Select** the breeding pool P of μ individuals from the previous generation. There are different strategies for selection and they depend on the cost. Individuals are more likely to join P if they are low cost.
 - For i in range(*eliteCount*, λ)
 - * Randomly choose two parents from P .
 - * Use **cross-over** to generate a child G_i^g . This means we take elements of each parent parameter set to make the child set.
 - * **Mutate** the values in G_i^g . This means we make small changes to each element of the child parameter set.
 - * Compute the child’s cost.
 - Sort the individuals in G^g by cost (in preparation for selection)

The key operators and parameters can be varied:

- Selection can favor fit (low-cost) parents more or less. If we favor them more, then the algorithm converges quickly, but it doesn't explore parameter space very well, and could miss a more fit individual. If we favor them less, the algorithm might not find children of increasingly good fitness. Explanations of several selection operators may be found in: Blickle, T. and Thiele, L. 1995. A comparison of selection schemes used in genetic algorithms. Tech. Rep. TIK-Report 11, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, May.
- Crossover is typically "uniform" (each element of the child is randomly taken from one of its two parents) or "single" (a cross-over point is selected and all elements up to the cross-over point are taken from one parent and the remaining elements are taken from the second parent).
- Mutation is typically implemented by choosing a value from a Gaussian distribution centered on each element. One straightforward way to vary the mutation is to vary the width of the distribution. We control the size of the mutation by scaling the width of the Gaussian distribution (it should be $mutationScale \cdot parameterValue$).
- The number of parents μ and children λ affects how widely we can sample space. Large numbers of children allow for a broader sampling. The number of parents (the size of the breeding pool) should be a fraction of the number of children (typically, it is a number like 1/5). It works with the selection operator to control how broadly the space is searched and how quickly the algorithm progresses towards a solution. The number of children that you need to use depends upon the problem. It can range from in the tens (e.g. a simple cost function we will use in class) to thousands.
- The number of elite individuals should be small (e.g. 1 to 3). Including elites guarantees that the best-fit individual in generation g will be at least as fit as the best-fit individual in generation $g - 1$.
- The number of generations should be as big as it needs to be for the best (or average) cost to stop improving. For some problems, thousands of generations are used. Here, we are able to use many fewer (e.g. I used 5 for the simple cost function in class).

We illustrate the algorithm with a cost function that searches for (1,5,2.5,2,4,3,0) in a space where every parameter is between 0 and 5. See Figures 1 and 2 for a description of the graphic.

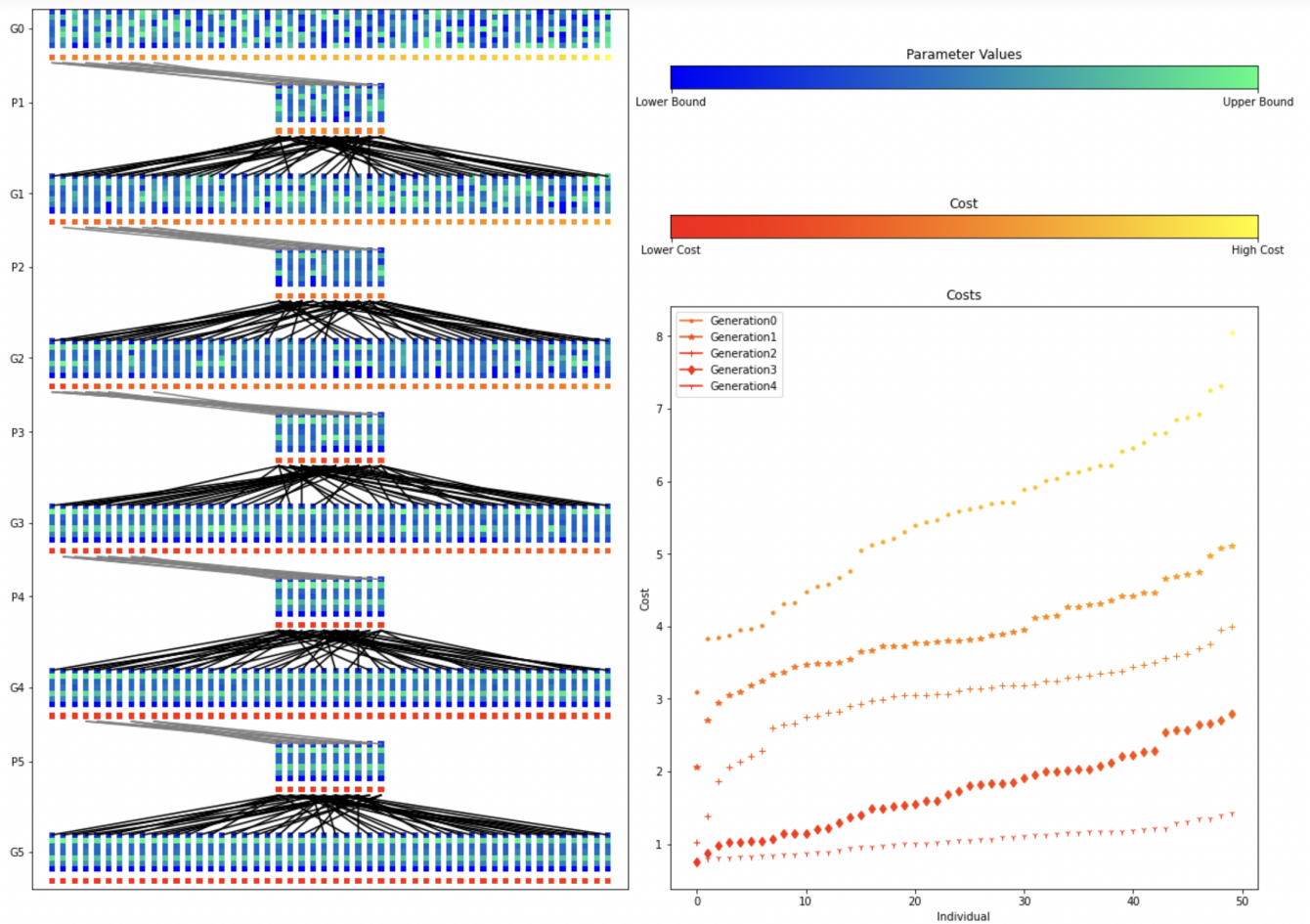


FIGURE 1. Genetic algorithm searching for $(1,5,2.5,2,4,3,0)$ in the hypercube with side-length 5 (i.e. all parameter values are between 0 and 5). The algorithm uses truncation selection to generate the breeding pool for each generation. Truncation selection allows only the best 1/5th of the individuals in the previous generation to be parents in the current generation. In the version of truncation selection that we are using, each member of the breeding pool is selected randomly from the best 1/5th (with replacement). This means only the very best individuals produce each generation and the algorithm improves the costs fairly rapidly. On the left, we represent each individual as a stack of 7 bluish colors. The color indicates the value of each parameter (with 0 as blue and 5 as green). Below the parameter value is the cost, also represented by color (orange is low cost and yellow is higher cost). Each generate G has 50 individuals. The individuals in generation G0 are chosen with random values between 0 and 5. Each subsequent generation has a breeding pool P1, P2, etc. with 10 individuals (selected using truncation selection). Lines drawn from the previous generation to individuals in the pool indicate which individuals were chosen. For every child in the current generation, two members of the pool are chosen randomly, and their parameters are combined (with uniform cross-over) to form the child, then each parameter is mutated slightly. Its cost is evaluated and shown in orange. All generations and breeding pools are shown with cost sorted from lowest (left) to highest (right). On the upper right, we show the colorbars for the parameters and costs. On the lower right, we show just the cost of each individual. It is color-coded as on the left and the marker shape indicates which generation it is part of.

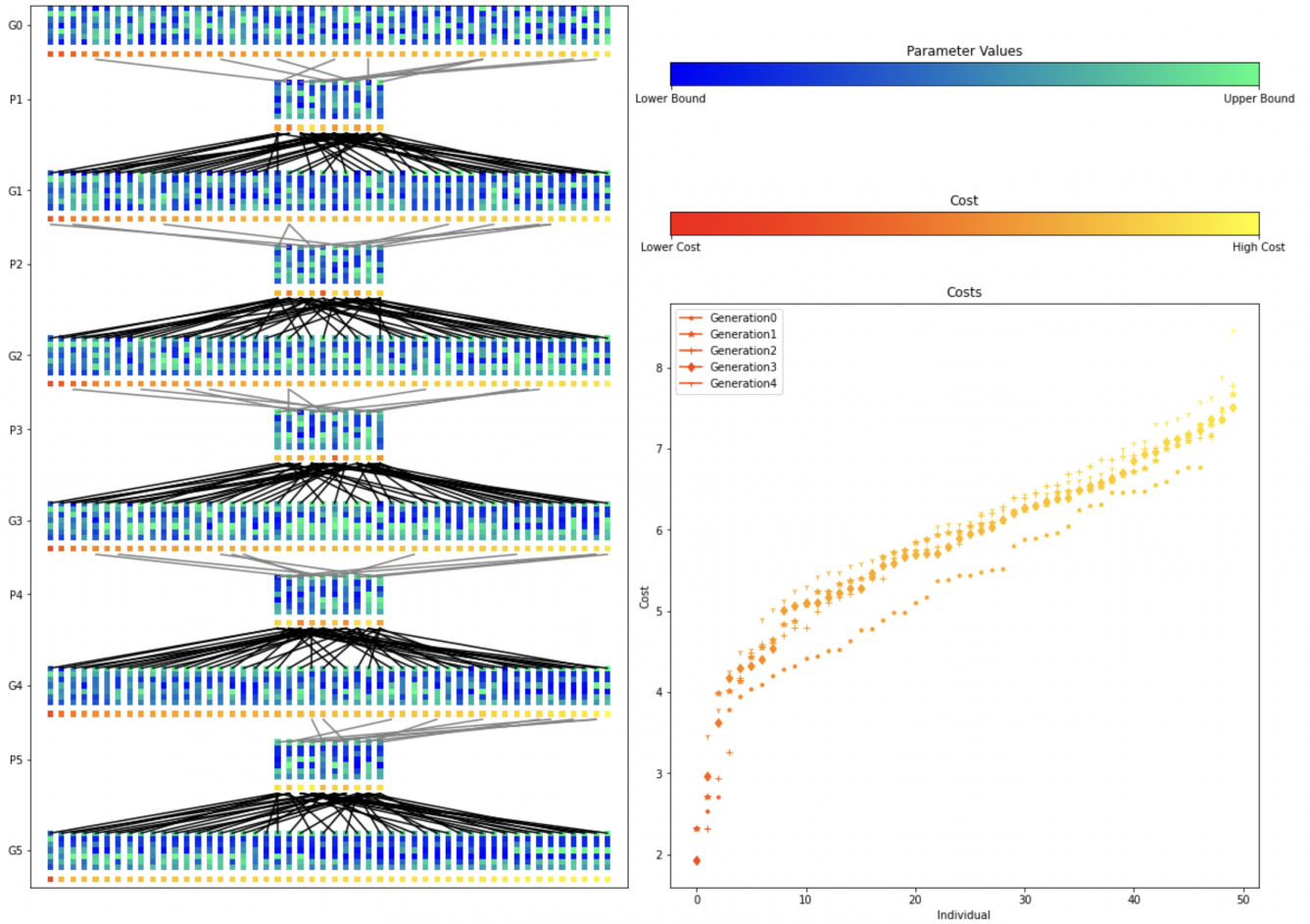


FIGURE 2. Genetic algorithm searching for $(1,5,2.5,2,4,3,0)$ in the hypercube with side-length 5 (i.e. all parameter values are between 0 and 5). The algorithm uses uniform selection to generate the breeding pool for each generation. Uniform selection applies no pressure. Therefore, the population does not reduce its cost from generation to generation. On the left, we represent each individual as a stack of 7 bluish colors. The color indicates the value of each parameter (with 0 as blue and 5 as green). Below the parameter value is the cost, also represented by color (orange is low cost and yellow is higher cost). Each generate G has 50 individuals. The individuals in generation G0 are chosen with random values between 0 and 5. Each subsequent generation has a breeding pool P1, P2, etc. with 10 individuals (selected using truncation selection). Lines drawn from the previous generation to individuals in the pool indicate which individuals were chosen. For every child in the current generation, two members of the pool are chosen randomly, and their parameters are combined (with uniform cross-over) to form the child, then each parameter is mutated slightly. Its cost is evaluated and shown in orange. All generations and breeding pools are shown with cost sorted from lowest (left) to highest (right). On the upper right, we show the colorbars for the parameters and costs. On the lower right, we show just the cost of each individual. It is color-coded as on the left and the marker shape indicates which generation it is part of.