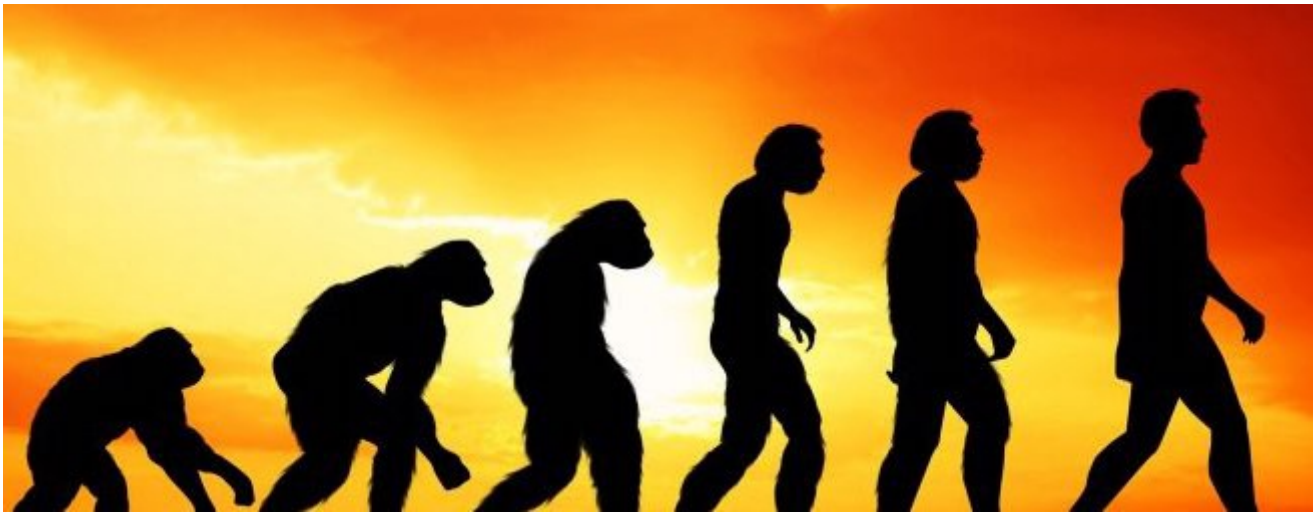
Introduction to Evolutionary Algorithms

Optimization by natural selection

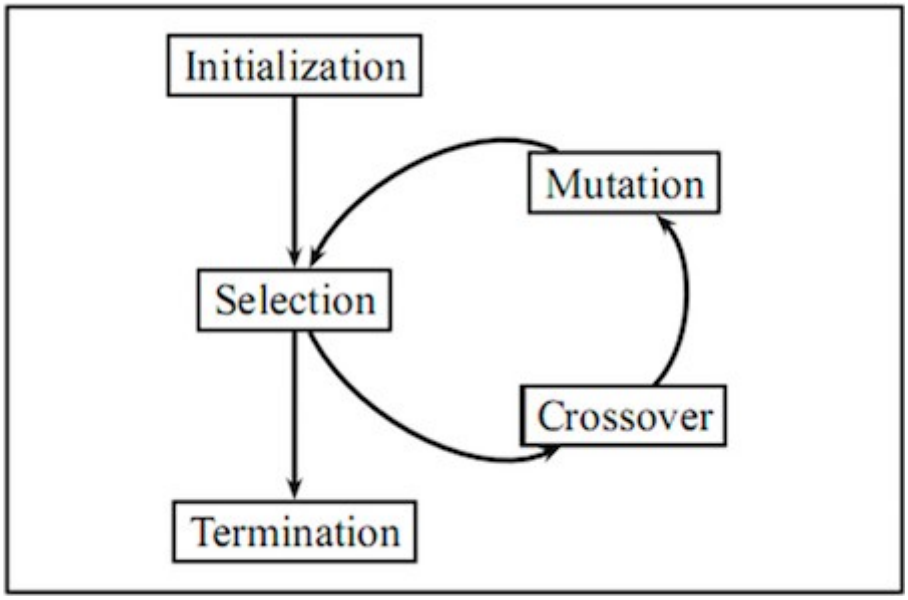
Devin Soni
Feb 18, 2018 · 4 min read ★



Evolutionary algorithms are a heuristic-based approach to solving problems that cannot be easily solved in polynomial time, such as classically NP-Hard problems, and anything else that would take far too long to exhaustively process. When used on their own, they are typically applied to combinatorial problems; however, genetic algorithms are often used in tandem with other methods, acting as a quick way to find a somewhat optimal starting place for another algorithm to work off of.

The premise of an evolutionary algorithm (to be further known as an **EA**) is quite simple given that you are familiar with the process of natural selection. An EA contains four overall steps: **initialization, selection, genetic operators, and termination**. These steps each correspond, roughly, to a particular facet of natural selection, and provide easy ways to modularize implementations of this algorithm category. Simply put, in an EA, fitter members will survive and proliferate, while unfit members will die

off and not contribute to the gene pool of further generations, much like in natural selection.



. . .

Context

In the scope of this article, we will generally define the problem as such: we wish to find the best combination of elements that maximizes some fitness function, and we will accept a final solution once we have either ran the algorithm for some maximum number of iterations, or we have reached some fitness threshold. This scenario is clearly not the only way to use an EA, but it does encompass many common applications in the discrete case.

Initialization

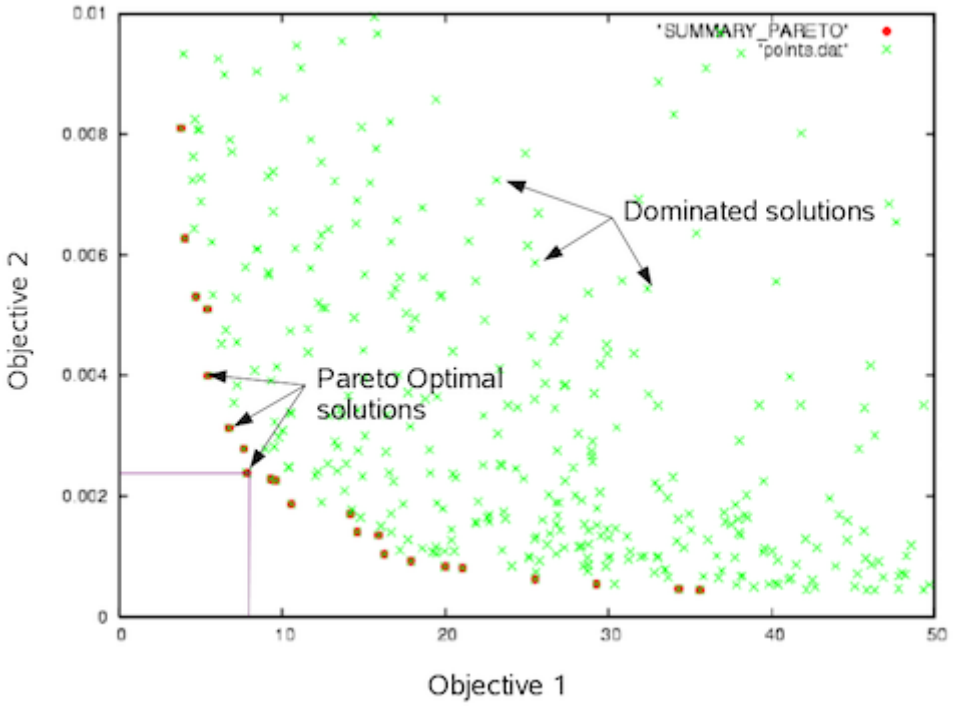
In order to begin our algorithm, we must first create an initial **population** of solutions. The population will contain an arbitrary number of possible solutions to the problem, oftentimes called **members**. It will often be created randomly (within the constraints of the problem) or, if some prior knowledge of the task is known, roughly centered around what is believed to be ideal. It is important that the population encompasses a wide range of solutions, because it essentially represents a gene pool; ergo, if we wish to explore many different possibilities over the course of the algorithm, we should aim to have many different genes present.

Selection

Once a population is created, members of the population must now be evaluated according to a **fitness function**. A fitness function is a function that takes in the characteristics of a member, and outputs a numerical representation of how viable of a solution it is. Creating the fitness function can often be very difficult, and it is important to find a good function that accurately represents the data; it is very problem-specific. Now, we calculate the fitness of all members, and select a portion of the top-scoring members.

Multiple objective functions

EAs can also be extended to use multiple fitness functions. This complicates the process somewhat, because instead of being able to identify a single optimal point, we instead end up with a set of optimal points when using multiple fitness functions. The set of optimal solutions is called the **Pareto frontier**, and contains elements that are equally optimal in the sense that no solution dominates any other solution in the frontier. A **decider** is then used to narrow the set down a single solution, based on the context of the problem or some other metric.



Genetic Operators

This step really includes two sub-steps: **crossover and mutation**. After selecting the top members (typically top 2, but this number can vary), these members are now used to create the next generation in the algorithm. Using the characteristics of the selected

parents, new children are created that are a mixture of the parents' qualities. Doing this can often be difficult depending on the type of data, but typically in combinatorial problems, it is possible to mix combinations and output valid combinations from these inputs. Now, we must introduce **new genetic material** into the generation. If we do not do this crucial step, we will become stuck in local extrema very quickly, and will not obtain optimal results. This step is mutation, and we do this, quite simply, by changing a small portion of the children such that they no longer perfectly mirror subsets of the parents' genes. Mutation typically occurs probabilistically, in that the chance of a child receiving a mutation as well as the severity of the mutation are governed by a probability distribution.

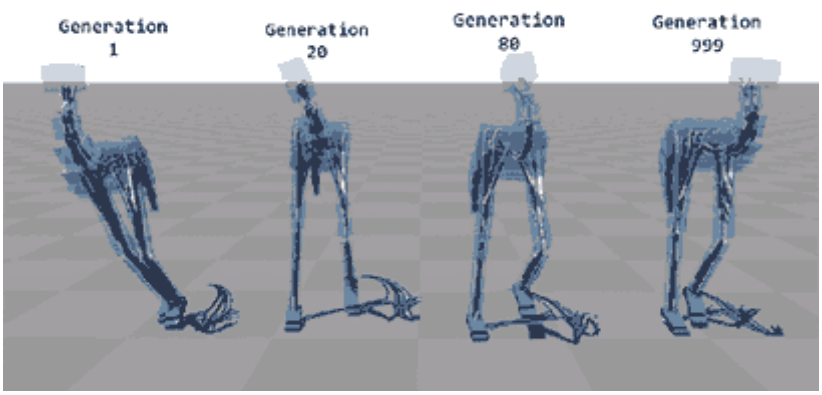
Termination

Eventually, the algorithm must end. There are two cases in which this usually occurs: either the algorithm has reached some **maximum runtime**, or the algorithm has reached some **threshold of performance**. At this point a final solution is selected and returned.

. . .

Example

Now, just to illustrate the result of this process I will show an example of an EA in action. The following gif shows several generations of dinosaurs learning to walk by optimizing their body structure and applied muscular forces. From left to right the generation increases, so the further right, the more optimized the walking process is. Despite the fact that the early generation dinosaurs were unable to walk, the EA was able to evolve the dinosaurs over time through mutation and crossover into a form that was able to walk.



Optimization Data Science Machine Learning Computer Science Towards Data Science

About Help Legal