Evolutionary Algorithms

Presented by
Muhannad Harrim

Overview

- This presentation will provide an overview of evolutionary computation, and describe several evolutionary algorithms that are currently of interest.
- Important similarities and differences are noted upon all the distinct themes of the evolutionary algorithms which lead to a discussion of important issues that need to be resolved, and items for future research.

Introduction

- Evolutionary computation uses the computational model of evolutionary processes as key elements in the design and implementation of computer-based systems and problem solving applications.
- There are a variety of evolutionary computational models that have been proposed and studied which we will refer to as evolutionary algorithms.
- They share a common conceptual base of simulating the evolution of individual structures via processes of selection and reproduction.
- They depend on the performance (fitness) of the individual structures.

Evolutionary algorithms (EA)

- More precisely, evolutionary algorithms maintain a population of structures that evolve according to rules of selection and other operators, such as recombination and mutation.
- Each individual in the population receives a measure of its fitness in the environment.
- Selection focuses attention on high fitness individuals, thus exploiting the available fitness information.

Evolutionary algorithms (EA)

- Recombination and mutation perturb those individuals, providing general heuristics for exploration.

- Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

Evolutionary algorithms (EA)

- A population of individual structures is initialized and then evolved from generation to generation by repeated applications of evaluation, selection, recombination, and mutation.

- The population size N is generally constant in an evolutionary algorithm.

Evolutionary algorithms (EA)

```
procedure EA
{
    t = 0;
    initialize population P(t);
    evaluate P(t);
    until (done) {
  t = t + 1;
  parent_selection P(t);
  recombine P(t);
  mutate P(t);
  evaluate P(t);
  survive P(t);
   }
}
```

Evolutionary algorithms (EA)

- An evolutionary algorithm typically initializes its population randomly, although domain specific knowledge can also be used to bias the search.
- Evaluation measures the fitness of each individual according to its worth in some environment.
- Evaluation may be as simple as computing a fitness function or as complex as running an elaborate simulation.

Evolutionary algorithms (EA)

- Selection is often performed in two steps, parent selection and survival.
- Parent selection decides who becomes parents and how many children the parents have.
- Children are created via recombination, which exchanges information between parents, and mutation, which further perturbs the children.
- The children are then evaluated. Finally, the survival step decides who survives in the population.

Evolutionary algorithms (EA)

- The origins of evolutionary algorithms can be traced to at least the 1950's.

- three methodologies that have emerged in the last few decades:
  - "evolutionary programming" (Fogel et al., 1966)
  - "evolution strategies" (Rechenberg, 1973)
  - "genetic algorithms" and "genetic programming" (Holland, 1975).

Evolutionary algorithms (EA)

- Although similar at the highest level, each of these varieties implements an evolutionary algorithm in a different manner.

- The differences include almost all aspects of evolutionary algorithms, including the choices of representation for the individual structures, types of selection mechanism used, forms of genetic operators, and measures of performance.

Evolutionary programming (EP)

- developed by Fogel (1966), and traditionally has used representations that are tailored to the problem domain.
- For example, in real-valued optimization problems, the individuals within the population are real-valued vectors.
- Other representations such as ordered lists, and graphical representations could be applied depending on the problem itself.

Evolutionary programming (EP)

```
procedure EP
{
  t = 0;
  initialize population P(t);
  evaluate P(t);
  until (done) {
        t = t + 1;
       parent_selection P(t);
       mutate P(t);
       evaluate P(t);
       survive P(t);
  }
}
```

Evolutionary programming (EP)

- After initialization, all N individuals are selected to be parents, and then are mutated, producing N children.
- These children are evaluated and N survivors are chosen from the 2N individuals, using a probabilistic function based on fitness.
- In other words, individuals with a greater fitness have a higher chance of survival.
- The form of mutation is based on the representation used.

Evolutionary programming (EP)

- For example, when using a real-valued vector, each variable within an individual may have an adaptive mutation rate that is normally distributed with a zero expectation.

- Recombination is not generally performed since the forms of mutation used are quite flexible and can produce perturbations similar to recombination, if desired.

Evolution strategies (ES)

- were independently developed by Rechenberg, with selection, mutation, and a population of size one.
- Schwefel introduced recombination and populations with more than one individual, and provided a nice comparison of ESs with more traditional optimization techniques.
- Evolution strategies typically use real-valued vector representations.

Evolution strategies (ES)

```
procedure ES; {
 t = 0;
 initialize population P(t);
 evaluate P(t);
 until (done) {
      t = t + 1;
      parent_selection P(t);
      recombine P(t)
      mutate P(t);
      evaluate P(t);
      survive P(t);
 }
 }
```

Evolution strategies (ES)

- After initialization and evaluation, individuals are selected uniformly Randomly to be parents.
- In the standard recombinative ES, pairs of parents produces children via recombination, which are further perturbed via mutation.
- The number of children created is greater than N.
- Survival is deterministic and is implemented in one of two ways:
  - The first allows the N best children to survive, and replaces the parents with these children.
  - The second allows the N best children and parents to survive.

Evolution strategies (ES)

- Like EP, considerable effort has focused on adapting mutation as the algorithm runs by allowing each variable within an individual to have an adaptive mutation rate that is normally distributed with a zero expectation.

- Unlike EP, however, recombination does play an important role in evolution strategies, especially in adapting mutation.

Genetic algorithms (GA)

- developed by Holland (1975), have traditionally used a more domain independent representation, namely, bit-strings.

- However, many recent applications of GAs have focused on other representations, such as graphs (neural networks), Lisp expressions, ordered lists, and real-valued vectors.

Genetic algorithms (GA)

- procedure GA {

```
t = 0;
initialize population P(t);
evaluate P(t);
until (done) {
        t = t + 1;
        parent_selection P(t);
        recombine P(t)
        mutate P(t);
        evaluate P(t);
        survive P(t);
    }
}
```

Genetic algorithms (GA)

- After initialization parents are selected according to a probabilistic function based on relative fitness.
- In other words, those individuals with higher relative fitness are more likely to be selected as parents.
- N children are created via recombination from the N parents.
- The N children are mutated and survive, replacing the N parents in the population.
- It is interesting to note that the relative emphasis on mutation and crossover is opposite to that in EP.

Genetic algorithms (GA)

- In a GA, mutation flips bits with some small probability, and is often considered to be a background operator.
- Recombination, on the other hand, is emphasized as the primary search operator.
- GAs are often used as optimizers, although some researchers emphasize its general adaptive capabilities (De Jong, 1992).

Variations on EP, ES, and GA Themes

- These three approaches (EP, ES, and GA) have served to inspire an increasing amount of research on and development of new forms of evolutionary algorithms for use in specific problem solving contexts.

Variations on EP, ES, and GA Themes

- One of the most active areas of application of evolutionary algorithms is in solving complex function and combinatorial optimization problems.

- A variety of features are typically added to EAs in this context to improve both the speed and the precision of the results.

Variations on EP, ES, and GA Themes

- A second active area of application of EAs is in the design of robust rule learning systems.
- Holland's (1986) classifier systems were some of the early examples.

Variations on EP, ES, and GA Themes

- More recent examples include the SAMUEL system developed by Grefenstette (1989), the GABIL system of De Jong and Spears (1991), and the GIL system of Janikow (1991).

- In each case, significant adaptations to the basic EAs have been made in order to effectively represent, evaluate, and evolve appropriate rule sets as defined by the environment.

Variations on EP, ES, and GA Themes

- One of the most fascinating recent developments is the use of EAs to evolve more complex structures such as neural networks and Lisp code.

- This has been dubbed "genetic programming", and is exemplified by the work of de Garis (1990), Fujiko and Dickinson (1987), and Koza (1991).

- de Garis evolves weights in neural networks, in an attempt to build complex behavior.

Variations on EP, ES, and GA Themes

- Fujiko and Dickinson evolved Lisp expressions to solve other problems.
- Koza also represents individuals using Lisp expressions and has solved a large number of optimization and machine learning tasks.
- One of the open questions here is precisely what changes to EAs need to be made in order to efficiently evolve such complex structures.

Representation

- Of course, any genetic operator such as mutation and recombination must be defined with a particular individual representation in mind.
- Again, the EA community differs widely in the representations used.
- Traditionally, GAs use bit strings. In theory, this representation makes the GA more problem independent, because once a bit string representation is found, standard bit-level mutation and recombination can often be used.
- We can also see this as a more genotypic level of representation, since the individual is in some sense encoded in the bit string.

Representation

- However, the GA community has investigated more distinct representations, including vectors of real values (Davis, 1989), ordered lists (Whitley et al., 1989), neural networks (Harp et. al, 1991), and Lisp expressions (Koza, 1991).

- For each of these representations, special mutation and recombination operators are introduced.

Representation

- The EP and ES communities are similar in this regard.

- The ES and EP communities focus on real-valued vector representations, although the EP community has also used ordered list and finite state automata representations, as suggested by the domain of the problem.

Representation

- Although much has been done experimentally, very little has been said theoretically that helps one choose good representations, nor that explains what it means to have a good representation.
- Messy GAs, DPE, and Delta coding all attempt to manipulate the granularity of the representation, thus focusing search at the appropriate level.
- Despite some initial success in this area, it is clear that much more work needs to be done.

Adaptive EA

- Despite some work on adapting representation, mutation, and recombination within evolutionary algorithms, very little has been accomplished with respect to the adaptation of population sizes and selection mechanisms.

- One way to characterize selection is by the strength of the selection mechanism.

- Strong selection refers to a selection mechanism that concentrates quickly on the best individuals, while weaker selection mechanisms allow poor individuals to survive (and produce children) for a longer period of time.

Adaptive EA

- Similarly, the population can be thought of as having a certain carrying capacity, which refers to the amount of information that the population can usefully maintain.
- A small population has less carrying capacity, which is usually adequate for simple problems.
- Larger populations, with larger carrying capacities, are often better for more difficult problems.

Performance Measures, EA-Hardness, and Evolvability

- Of course, one can not refer to adaptation without having a performance goal in mind.
- EA usually have optimization for a goal.
- In other words, they are typically most interested in finding the best solution as quickly as possible.

Performance Measures, EA-Hardness, and Evolvability

- There is very little theory indicating how well EAs will perform optimization tasks.
- Instead, theory concentrates on what is referred to as accumulated payoff.

Performance Measures, EA-Hardness, and Evolvability

- The difference can be illustrated by considering financial investment planning over a period of time (stock market).
- Instead of trying to find the best stock, you are trying to maximize your returns as the various stocks are sampled.
- Clearly the two goals are somewhat different, and maximizing the return may or may not also be a good heuristic for finding the best stock.
- This difference in emphasis has implications in how an EA practitioner measures performance, which leads to further implications for how adaptation is accomplished.

Performance Measures, EA-Hardness, and Evolvability

- This difference also colors much of the discussion concerning the issue of problem difficulty.
- The GA community refers to hard problems as GA-Hard.
- Since we are now in the broader context of EAs, let us refer to hard problems as EA-Hard.
- Often, a problem is considered difficult if the EA can not find the optimum.

Performance Measures, EA-Hardness, and Evolvability

- Although this is a quite reasonable definition, difficult problems are often constructed by taking advantage of the EA in such a way that selection deliberately leads the search away from the optimum.
- Such problems are called deceptive.
- From a function optimization point of view, the problem is indeed deceptive, however, the EA may maximize accumulated payoff.

Performance Measures, EA-Hardness, and Evolvability

- Another issue is also very related to a concern of De Garis, which he refers to as evolvability.
- De Garis notes that often his systems do not evolve at all, namely, that fitness does not increase over time.
- The reasons for this are not clear and remain an important research topic.

Distributed EA

- Recent work has concentrated on the implementation of EAs on parallel machines.
- Typically either one processor holds one individual (in SIMD machines), or a subpopulation (in MIMD machines).
- Clearly, such implementations hold promise of execution time decreases.

Summary

- Genetic algorithm - This is the most popular type of EA. One seeks the solution of a problem in the form of strings of numbers (traditionally binary, although the best representations are usually those that reflect something about the problem being solved - these are not normally binary), virtually always applying recombination operators in addition to selection and mutation.

- This type of EA is often used in optimization problems.

- It is very important to note, however, that while evolution can be considered to approach an optimum in computer science terms, actual biological evolution does not seek an optimum.

Summary

- Evolutionary programming - Like genetic programming, only the structure of the program is fixed and its numerical parameters are allowed to evolve, and Its main variation operator is mutation.
- Evolution strategy - Works with vectors of real numbers as representations of solutions, and typically uses self-adaptive mutation rates, as well as recombination.
- Genetic programming - Here the solutions are in the form of computer programs, and their fitness is determined by their ability to solve a computational problem.