CS B551

Elements of Artificial Intelligence

Homework 6: Genetic Algorithms for Local Search/Learning

Due Tues, Dec 8, at 4:00pm.

Introduction

The Knapsack problem is a famous, NP-Hard optimization problem. While it has been lucidly described in many sources, like its wikipedia page, there are several variants of the problem, so this assignment specifies exactly what we mean by the Knapsack problem. Given a volume ***V*** and a set of (volume, price) tuples ***C***, compute:

$$argmax_{B \subseteq C} \sum B.price \ \ such\ that \sum B.volume \ < V$$

Due to the NP-Hardness of the problem and the simplicity of encoding a chromosome and computing its fitness, this problem is a good candidate for genetic algorithms. People have given thought to this approach before, and have written tutorials you might want to look at.

Expectations

Fill out the stubs of the unimplemented functions in the support code:

- **reproduce(mom,dad)**
- **randomSelection(population,fitnesses)**
- **fitness(max_volume,volumes,prices)**
- **mutate(child)**

and

- **genetic_algorithm(world,popsize,max_years,mutation_probability)**

which uses all of the above appropriately. In the genetic algorithm, world is a data structure describing the problem to be solved. As illustrated in the provided code, problem instances passed to genetic algorithm are of the form:

```
instance = (max_volume, list_of_volumes, list_of_prices).
```

list_of_volumes and list_of_prices are zippable, so the i'th price and i'th volume are of the same item.

When you submit your assignment, include your code and the output of executing the **run** function, and the answer to the question below.

Your algorithm is not required to come up with the optimal solution because genetic algorithms are not designed to come up with an optimal solution. Instead, they are a robust method for achieving good solutions when little knowledge is available to guide search.

You may want to experiment with techniques like elitism and other techniques mentioned in the tutorial. You are encouraged to measure the effects of parameters like initial population size and mutation probability over a range of settings, and will submit a brief discussion for a few examples. In addition to ad-hoc ways to do so, you may want to modify the structure of table, to keep track of information about what high scores correspond to what parameter values, and do queries on the information you collect. Doing so would be a good opportunity to get a feel for genetic algorithms. It is also an opportunity to learn to use python's pandas library, which is an excellent technology for handling tabular data (it handles things like interfacing with SQL databases and HDF5, supports rich queries, unknown information, and plays nicely with visualization libraries like seaborn).

The Support Code

You can download support code at http://www.cs.indiana.edu/classes/b551/knapsack.py. Note that there is no "main" function in the support code, so that you will need to write your own to call "run" with the parameters of your choice.

Important: There are two potentially confusing aspects of the support code. First, in order for your **genetic_algorithm** implementation to correctly interact with the data collection done in **run**, you should accumulate a (plans, fitness of plans) tuple for each year, and return it as a list of tuples. Second, if you decide to use the **compute_fitnesses** function, **chromosomes** is required to be a 2D numPy array of boolean values (a fixed-size boolean array is the recommended encoding of a chromosome, and each array might be thought of as a row, hence the matrix).

Questions

If you have confusions about the support code or about the interfaces provided by the function stubs, please post them to the **canvas discussion board** for this assignment, so that everyone in the class will be able to benefit from the answers. For support code questions not relevant to the class as a whole, please contact Alex at aseewald@indiana.edu. The instructor and all AIs are happy to help with any questions on genetic algorithms, etc.

Submission

Please submit using canvas. In addition to your code, please submit a PDF file with your name, answering the following question:

What parameter settings gave you what you consider the best performance by the GA, for the easy and hard problems? Compare to two other parameter settings you've tried for the GA for the knapsack problem. Compare the behavior of the GA for those settings, in terms of how those settings affect convergence behavior and final solution quality. Explain these behaviors in terms of general properties of GAs. If any results were surprising, try to explain why they might have occurred.

Please let us know if you have any questions!