A Description of the necessary components for a parallel info econ simulator.

In this document, I’ll describe the existing code for conducting infoecon simulations, what each of the components do, and what the new, parallelized simulator should be able to do. I’ll begin giving an overview of the simulation, then I’ll describe each of the classes in the existing code and what they do.

1 Overview

The simulation is a fairly standard agent-based simulation of an economy. This economy consists of one or more producers, who offer bundles of information goods to a consumer population. Each consumer has a valuation on these goods; this consists of an amount they’re willing to pay for their favorite type of good, the number of goods they value, and a clutter cost. The consumer population is typically heterogeneous; this means that even though an individual consumer is relatively simple, the aggregate population is complex, and the mapping from offered bundles to profit is highly nonlinear.

So far, experiments have focused on the producers’ side of the problem. Producers start out with no knowledge of consumer preferences, and must learn profitable bundles and/or prices through trial and error. This leads to an exploration/exploitation tradeoff; by trying new bundles, a producer might learn more, but at the cost of current short-run profits.

This sort of simulation environment can be used to examine a number of different questions. For example, classical microeconomic theory predicts that producers with complete knowledge will often compete directly with each other through sequential undercutting, even though they might gain more profit by playing “less intelligently.” Another related question that can be examined involves the formation of niche markets; how can producers who are independently searching for profitable locations in “product space” segment the market in a self-organized fashion so that consumers are satisfied and producers extract significant profits. A third question involves the value of learning; when the consumer population is changing, a producer may be better off just “holding still rather than trying to learn or adapt. Characterizing when this is the case, and whether it depends upon the behavior of other producers, is an open problem.
2 Modeling Information Goods

We assume that information goods can belong to one of a set $C$ of categories (think news, sports, arts, etc. as categories.) Every article belongs to exactly one category. In the simplest formulation, these categories are all disjoint sets that can be arranged by similarity on a line. (I know, it’s not very realistic, but it’s a first crack.) More complex arrangements would include tree and poly-tree structures, subsumption and inheritance, and articles could belong to multiple categories.

Producers (see below) offer a bundle of goods, where a bundle consists of a vector indicating the number of articles in each category, along with a price schedule. Initially, the price schedule can consist of a price for the bundle - this lets us focus on the problem of selecting articles to offer and finesse the less-interesting price selection problem. In a lot of cases, when producers can only compete in terms of price, they cannot differentiate themselves, and the result is a fairly uninteresting price war.

So we will need to have a struct representing a bundle, which will consist of an $n+1$ dimensional vector, where $n$ is the number of categories. The first $n$ elements indicate the number of articles in each category, and the last element is the price.

3 A Consumer

In this section, I’ll describe the relevant characteristics (instance variables) and behavior (methods) of a consumer.

A consumer can be described by four parameters, $w, c^*, k$, and $\lambda$. $w$ indicates the value a consumer gives to its most-preferred category $c^*$. $k$ indicates the fraction of categories that a consumer values positively. The value of articles in non-favorite categories is given by:

$$V(c) = \begin{cases} 
  w(1 - \frac{2(|c^* - c|)}{kn}) & \text{if } |c^* - c| \geq kn \frac{2}{3} \\
  0 & \text{otherwise}
\end{cases}$$

This produces a tent-shaped distribution of values around the favorite category. The value of a bundle is then the sum of the value of each article, less the clutter cost. Clutter cost is a nonlinear function that models the cost a consumer incurs in consuming this bundle. (It can be thought of as bandwidth or storage, or as the value of time spent sorting through the bundle.) Clutter cost is denoted by the function $\alpha(|B|)$, where $|B|$ is the size of the bundle, and represented as: $\alpha(B) = e^{\lambda|B|}$. By changing $\lambda$, we can control the rate at which clutter cost begins to matter to consumers. On every iteration, each consumer will choose from amongst the offered bundles and select the one which maximizes its surplus (value - price). If no bundle produces positive surplus, the consumer buys nothing.

So, a consumer must be able to:
• Determine the value of a bundle (less clutter cost).

• Choose between bundles.

### 3.1 Generating consumers

In some experiments, we will want the consumer population to change over time (for example, unhappy consumers might drop out). It would be nice to have a separate object which manages the consumer population. It should be able to:

• Generate a population according to some distribution.

• Probabilistically select consumers to cull.

• Replace these consumers with new, random consumers.

### 4 Producers

As discussed above, producers are at the heart of the work to date. They choose bundles, offer them to consumers and learn from their past performance. A producer is interested in maximizing its aggregate profit; unfortunately, solving directly for the optimal set of actions to take is intractable, and so we use a greedy learning algorithm instead.

One particularly useful algorithm for noisy, nonstationary environments is a genetic algorithm. In this algorithm, bundles are encoded as bitstrings and then tested on the consumer population, where they receive a fitness (the profit they make). Bundles (genes) are then probabilistically carried over to the next generation based on their fitness; a higher fitness has a more likely chance of surviving. New bundles are created using crossover. ¹

This will be the most involved part of the simulation. We will need a gene struct that is able to map bundles into bitstrings and vice versa. We will also need functions to perform selection, mutation and crossover. We can talk more about this as we go along.

### 5 The simulation engine

We’ll also need a piece of code that manages the whole simulation, including creating producers and consumers, calling their appropriate methods, keeping track of data, and so on. It may be helpful to have two processes: one which manages the “while loop” portion of the simulation, and one which hands out tasks to different processors. This second component would implement what is called the “bag of tasks” approach. For example, assume that all the producers have chosen bundles, and now each consumer must decide what to buy. Each consumer’s decision is a task, and so the “bag manager” would determine how

---

¹This is an exceedingly brief description of GAs; more detailed information can be found in Melanie Mitchell’s book.
many nodes are available for computation and assign each of these tasks to a node. If there are more tasks than nodes, then it is responsible for assigning new tasks as nodes finish their old tasks. Once all the computation is complete, it notifies the “while-loop manager” (a lame name, btw) that the simulation can proceed.

6 Other things

Other potential tools include the development of other learning algorithms, visualization tools for analyzing the data, and a scripting language to allow for the setup and automated running of many experiments with varied parameters.