Introduction to Artificial Intelligence
Program 2 – Simulated Annealing
Fall 2021

You're preparing for a back-country camping trip. Your campsite is very remote, so you want to be prepared and have everything with you that you will need.

BUT—your vehicle will only hold so much. There just isn't room for everything you might possibly want; you're going to have to make some choices. So you begin by making a list of anything you could think of that you might possibly want—you don't want to forget something simple. But how to decide what to take?

You have assigned each item a *utility*, a degree of usefulness, as a floating-point number in the range 0-10. An item rated 0 or 1 is something you could easily do without if necessary; something rated 9 or 10 is vital. And of course, each item weighs something—a real number > 0, in pounds. Your vehicle will carry 500 pounds of cargo. How can you pack the most useful items (highest utility) into the available space?

(We will assume that weight, not volume, is the limiting factor; no matter what combination of items you select, they will fit in the vehicle, as long as the total weight is less than 500 pounds. This is an example of the *knapsack problem*; finding an exact solution is NP-complete, meaning it takes exponential time; even if we could instantly guess the best combination, confirming that it really was the best solution would also take exponential time. A brute-force solution requires n! time, and a dynamic-programming solution requires $O(n^2 2^n)$ time. Still exponential, still not practical for any but small N. So we'll settle for a 'pretty good' solution we can find in a practical time frame.)

You are given an input text file containing 400 sets of (utility, weight) pairs. Your task is to use simulated annealing to find a good selection of items to pack while staying within your weight guidelines. Your program will be tested against a different data file with the same format.

Programming details:

- You may write your program in Python, C, C++, C#, or Java.
- The simplest way to represent a selection is as a bitstring or something that maps to one, with '0' indicating the item is not being selected and a '1' indicating that it is. This can be an array or vector of booleans, of characters, bytes, a string, or even literal packed bits, in which case you will need only 50 bytes to represent a selection (but be prepared for lots of low-level bit manipulation).
- Simulated annealing works best when trying to minimize a quantity, but we want to increase utility. So, we will pick an (arbitrary) value of 1000 and try to minimize the distance between that and the total utility of all items we have selected. That is, minimize (1000 – total utility of all items). Account for the weight limit by using the unmodified total if the total weight of all items is below 500, and a penalty of -20 utility for every pound over the weight limit.
- Initially select about 1/20 of the items (randomly) to include.
- In this case, a possible change to the selection is just to choose an item at random (uniform selection) and toggle its boolean value.
- The utility values will be rather high, so select a fairly large value for the initial temperature.
- Since the population is 400 items, reduce the temperature every 4000 successful changes or 40,000 attempts. Continue until an entire iteration (4000 changes / 40,000 attempts) passes with

no changes accepted. At that point, report the number of items packed, total utility, and total weight.

Prepare a short report—a page or two—describing your program. Include a discussion of what data structures you chose, and why; any places you were able to optimize your program, or there was a bit of coding you're particularly proud of; and an overview of what running the program was like—what was the overall efficiency like, how many iterations were needed, etc. No references are expected.

Submit your report along with your source code and output file to Canvas. If you are using GitHub or other online repository, submit your GitHub link.