Knapsack problem

The knapsack problem can be defined as follows. Given a set of items, each with a weight and a value, and given a capacity, determine the number of each item to include in a collection so that the associated total weight is less than or equal to the capacity, and so that the total value is as large as possible. The knapsack problem derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

This problem often arises as a subproblem in resource allocation applications where there are financial constraints, such as:

- Cargo loading (truck, boat, cargo aircraft)
- Satellite channel assignment
- Portfolio optimization.

Let us consider that we have n-objects that must be put in a bag with a weight limit noted P. Each object i has a value v_i and a weight w_i. We will consider the binary version of the knapsack problem for which we must decide whether we choose an object or not (no possibility of embarking multiple copies of the same item)

Problem setup:

Define the objects as a dictionary {object: (weight, value)}, for example:

Objects = {'A': (10, 2), 'B': (6, 3), 'C': (4, 8), 'D': (8, 5), 'E': (9, 5), 'F': (7, 6)}

Capacity of the bag: C = 15

Items = list(Objects.keys()) # Items is a list of all keys of dictionary Objects

Variable Items will be used to access the Objects dictionary. Items is a list ['A', 'B', 'C', 'D', 'E'].

State representation:

A state is represented as a binary list (either 0 or 1) for each item; 0 means item is left, and 1 means item is taken in the bag. For example, state = [0, 1, 1, 1, 0, 0]; this means that items B, C and D are taken into the bag.

Value function:

The value function takes a state as a parameter and will do two things:

1) Compute the total weights of items taken in the bag (items with 1 in the state list). If the total weight exceeds the capacity C, then returns a value of -1 indicating illegal state. Otherwise, go to step 2.
2) Compute the total value of items taken in the bag, return the total value.

Get-neighbor function

This get-neighbor function takes a state as a parameter. It will generate a new state by randomly selecting an item and flip its value ($1 \rightarrow 0$ and $0 \rightarrow 1$).

Write a program using simulated annealing to maximize the value of item picked in the bag that do not exceed the bag capacity. Use the template attached with the assignment.

What to submit:

- Create a GitHub repository and upload your program.
- Submit to Canvas the GitHub link to your repository.