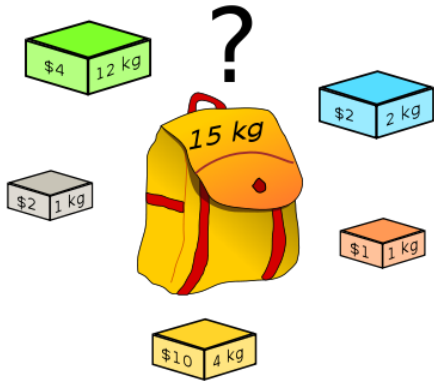


CSCU9YE - Artificial Intelligence

Lab 2: Combinatorial Optimisation - The Knapsack Problem

The Knapsack Problem

Consider the following *optimisation* problem. We are given a backpack or knapsack of a given maximum capacity W , and a number n of possible items to pack on it. Each item has a value and a weight associated to them. The goal is to fill the backpack with as many items as possible (i.e within the weight limit), while maximising the total value of the items packed. You can think of it as if you're preparing for a trip outdoors and you have to pack the items you need, but you do not want to exceed a given weight. This problem is known as the *Knapsack* problem. The data shown below represents an example (or instance) of the Knapsack problem (https://en.wikipedia.org/wiki/Knapsack_problem).

 <p>Image from Wikipedia</p>	<p>Dataset describing the example in the left. The first line gives the number of items, in this case 5. The last line gives the capacity of the knapsack, in this case 15. The remaining lines give the index, value and weight of each item.</p> <pre>5 1 2 1 2 4 12 3 2 2 4 1 1 5 10 4 15</pre>
----------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

You are given:

- The Python code to read a given dataset of the Knapsack problem: [knapsack.py](#)
- Datasets for knapsack instances with 20 and 200 items: [knap20.txt](#), [knap200.txt](#)

Notice that for your program to work your datafiles should be in the same directory of your script. Otherwise you will need to indicate or change the path. Here is an example of how to call the function to read the file with the knapsack data. In the program, the `with` statement ensures that any file opened in the statement is properly closed once the end of the block is reached.

```
# Reading data from a knapsack file
```

```
knapfile = 'knap20.txt'
nitems, cap, values, weights = read_kfile(knapfile)
```

Your task in this lab is to solve the Knapsack problem, using different solving strategies:

- A. A greedy constructive heuristic
- B. A random search heuristic
- C. Fully enumerating the search space (Optional)

Here a detailed description of your tasks:

1. Before starting to code your algorithms (heuristics), your first task is to visualise your dataset. Using matplotlib, produce a dot plot where the x axis contains the weights and the y axis the values of the values. Explore how to add labels to both axes
2. Implement a random search heuristic in a modular way. Following the guidelines below, implement:
 - A function called `def random_sol(n)`, which receives as input an integer number n (which needs to match the size of the instance) and returns a list of binary (1 and 0) numbers of length n , generated at random. For example, if $n = 5$, like in the instance above, a possible output is `[1, 1, 0, 1, 0]`. This list represents a set of items, where a '1' in a position i indicates that item i is in the backpack, and a '0' indicates that it is not. So for example the solution above, `[1, 1, 0, 1, 0]`, indicates that the 1st, 2nd and the 4th items are in the backpack.
 - A function called `def evaluate(sol)`, which receives as input a binary list `sol`, and returns two values, `total_val`, `total_wei`, corresponding to the total sum of values and weights of the items, respectively. For example, if we consider the small instance data shown in the previous page for $n = 5$, calling the function with `sol = [1, 1, 0, 1, 0]`, should return a total value of 7 and a total weight of 14.
 - A function called `def random_search(tries)`, which receives an integer number and returns a solution to the Knapsack problem. Using a loop the function generates `tries` number of random solutions using `random_sol(n)` and evaluates them using `evaluate(sol)`. The function should keep a record of the best valid solution generated in the loop. Notice that a valid solution should have a total weight which is less than or equal to the capacity of the knapsack. The best valid solution will be that obtaining the largest total value from all solutions generated. The function should return three outputs, namely, the best solution found (i.e the binary list), the solution value and the solution weight.
3. Implement a function called `def constructive()` which returns a solution to the Knapsack problem using a Greedy Construction Heuristic, following the pseudo-code discussed in the lecture.

Greedy Construction heuristic for Knapsack [heuristic]:

1. start with an empty knapsack
2. repeat until no more items can be added:
3. determine ~~i~~next the most valuable item that still fits.
4. add ~~i~~next to the knapsack

Some guidelines for implementing the constructive heuristic are as follows:

- Your function should return two values. First, a solution encoded as a list of item indices. For example if your solution contains items number 0, 7 and 12, your solution should be [0, 7, 12] (notice that list indices start with 0 in Python). Second, a numerical value representing the weight of your solution.
- Some useful predefined Python functions you may want to use in your implementation are:
 - `best = max(values)` # determine maximum value in a list
 - `i = values.index(best)` # determine index of a given value in a list
 - `del values[i]` # delete item in position 'i' from a list

4. **Optional:** If you have the time, and are curious, try implementing an exhaustive enumeration of the search space. Can you do it for $n = 20$? What about $n = 200$?

Show:

- Plot with weights vs. values with labels.
- Code and execution of function `def random_sol(n)` several values of n .
- Code and execution of function `def random_search(tries)`.
- Code an execution of your Greedy Construction Heuristic.
- What is the best solution you could find for this problem in this lab?
- Which of the two algorithms produced the best result?