# Knapsack1

Olof Harrysson

October 2016

## 1 Problem Statement

There exist an unknown number of items, each with it's own weight and cost. The problem consists of choosing items to maximize the combined cost whilst not exceeding a given weight.

n = number of items
w = weight of an item
c = cost of an item
M = maximum combined weight

n, w, c and M all have to be positive real numbers.

### 1.1 Resources

The algorithms were implemented in Python 3.5.1. Pythons method time() in the time module was used to measure elapsed time. The code was run on a 2,3 GHz Intel Core i7, MAC OSX Yosemite. The library Numpy was used to create the graphs presented.

## 2 Analysis of possible solutions

A possible solution to an instance of the problem is a combination of items that doesn't exceed the weight limit. There can be different combinations of items who yield the same maximum cost.

## 3 Algorithms

This paper will describe two ways to solve this problem.

### 3.1 Brute Force

The brute force algorithm is an algorithm which can be used for a multitude of purposes. It enumerates over all possible solutions until it has met a criteria or it has exhausted all combinations.

The brute force algorithm generally isn't very complicated. In this implementation of the brute force algorithm, it checks if a combination of items is to heavy and continues with other configurations instead of trying combinations that it knows will be to heavy.

### 3.2 Greedy

The greedy approach first sorted the items by their cost to weight ratio. It then added the item with the highest ratio if it didn't make the configuration exceed the weight limit. It's run time is O(n) which is fast but doesn't always produce the right solution.

## 4 Results

### 4.1 Brute Force

The brute force method produced the right solutions for all instances up until n=20. For higher values of n the algorithm didn't finish within two minute so the program was terminated.

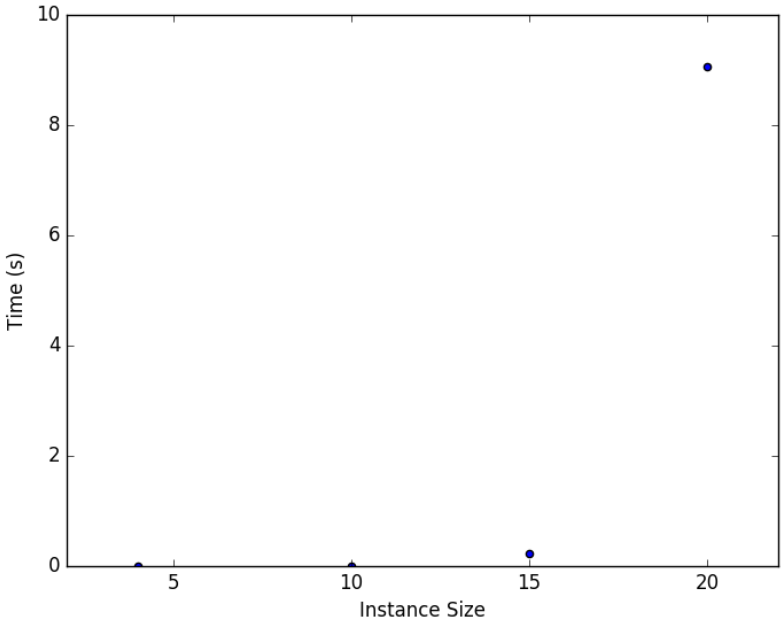


Figure 1: Time for brute force

## 4.2 Greedy

The greedy algorithm produces the wrong solution in up to approximately 60% for some instance sizes and increases as the instance size gets larger, see Figure 2. The relative error for these incorrectly solved solutions goes as high as 37% and tend to decrease as n gets larger.

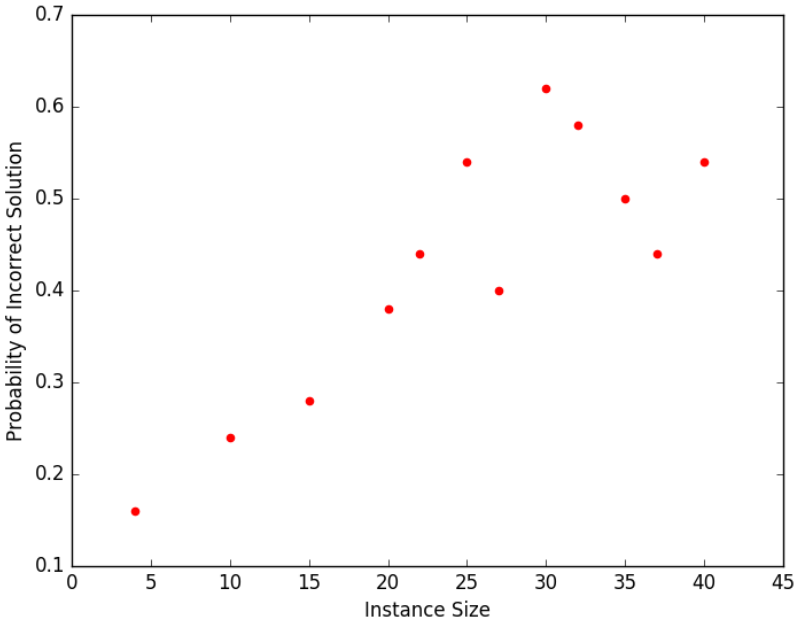


Figure 2: Probability of incorrect solutions using the greedy approach.

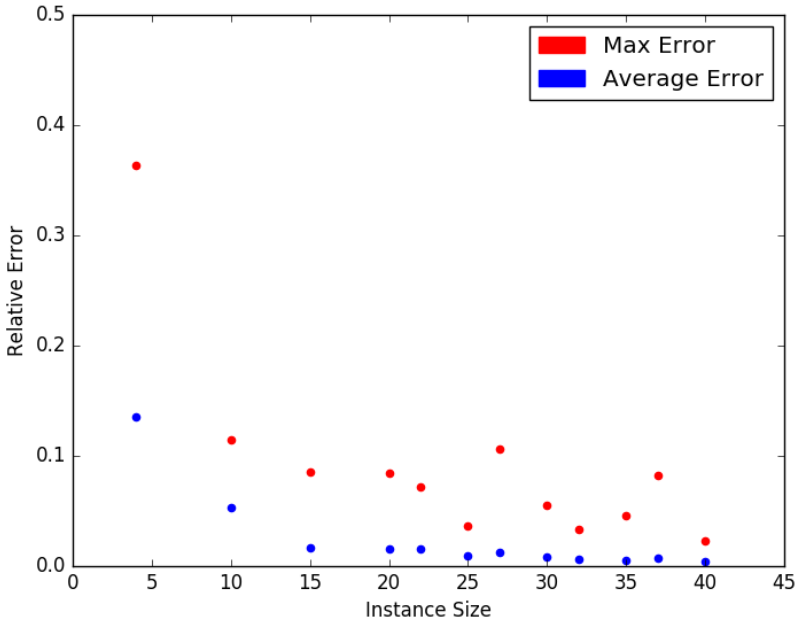


Figure 3: Relative Error for Greedy

As seen by Figure 3, the time complexity for the greedy approach is linear and completes in a fraction of a second even for n values as large as 40.

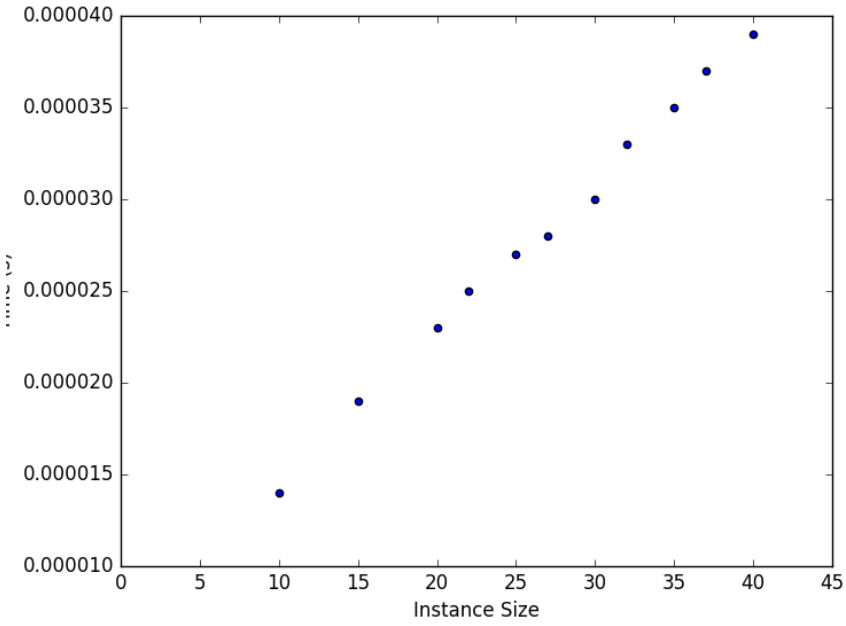


Figure 4: Time for brute force

# 5 Conclusions

As seen in Figure 1, the time complexity for the brute force algorithm is exponential. It is therefor usable if n is very large. It does however always produces the right result if it completes. Therefor the brute force is better for applications where time can be sacrificed and n is not very large.

The greedy algorithm is very fast even for larger instances as it follows a linear model. It is however producing the wrong solution a lot of the time. As n gets larger the relative error decreases for most instances. The greedy method might prove useful if the solution doesn't need to be exact.