COMPARISON OF HEURISTIC ALGORITHMS FOR KP01

A comparison of heuristic algorithms for solving the 0-1 Knapsack Problem

Raunak Redkar

Supervisors: Per-Olof Freerks and Felicia Dinnetz

Kungsholmens Gymnasium

Contents

§1. Introduction

§1.1 Background

There are many situations in every day life, where a person wonders whether he is doing something efficiently. Unforunately, the human brain is not always capable of coming up with an optimal approach when the problem has a lot of factors at play. It is here when a system is used, and the true power of computing can be recognised.

To solve such problems, a computer is provided all the information, from which it will produce an optimal answer. For the system to process all the information, certain instructions must be written into the system so that it knows how to handle the information. This set of instructions is called an algorithm. The system or computer uses algorithms to take in information (the input), and produce an answer - an output. The efficiency (in all aspects) of the algorithm is dependant on the instructions which make up it. There are many algorithms which have been designed to solve many problems. For problems in computer science, more than one algorithm is usually proposed and used. Optimization problems are where this is frequently the case.

In the fields of computer science and mathematics, optimization problems are problems of finding the best solution, from a range of many feasible solutions. They are usually categorized into 2 categories: discrete optimizations and continuous optimizations, depending on whether the variables are discrete or continuous respectively. Combinatorial optimization problems are a subset of optimization problems that fall into the discrete. Combinatorial optimization involves searching for a maxima or minima for an objective function whose search space domain is a discrete but (usually large) space.

Typical combinatorial optimization problems are not limited to but include:

- General Knapsack Problem - Given a set of items, each with weight and profit value and a knapsack capacity, what is the best way to choose the items while respecting the knapsack capacity?
- Traveling Salesman Problem- Given a list of cities, what is the shortest possible path that visits each city exactly once and returns to the origin?
- Set Cover - Given a set of elements $\{1, 2, ..., n\}$, what is the and a collection of m sets whose union equals the universe, what is the smallest sub-collection of sets whose union is the universe?

Combinatorial optimization problems show up in an array of different fields. The Knapsack Problem in particular has many variants which include the 0-1 knapsack problem, the bounded and unbounded knapsack problems, the multidimensional knapsack problem, the discounted knapsack problem, etc. The 0-1 Knapsack Problem is the simplest form of the knapsack problem and thus has also been the main focus in the research community. It appears in real-world decision-making processes in a variety of fields. Some examples include:

Many of combinatorial optimization problems including the 0-1 Knapsack problem currently do not have deterministic algorithms which are considered fast enough for them to be used on a large-scale basis. Consequently, the research focus has been on approaches that do not necessarily guarantee the best solution but win over deterministic approaches when it comes to time.

Problem statement

In the 0-1 Knapsack Problem, there n items and a maximum weight capacity W. Each item has a profit value p_i and a weight value w_i. Find the optimal selection of items which maximizes the profit value while respecting the max weight value. The problem can be mathematically represented as so:

$$\text{Maximize } f(\vec{x}) = \sum_{i=1}^{n} p_i x_i \tag{1}$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i <= W \tag{2}$$

$$x \in \{0, 1\} \tag{3}$$

The rest of this paper will follow the shorthand "KP01" for the 0-1 Knapsack Problem.

§1.2 Aim

The aim of this paper is to compare various algorithms in order to investigate the strength of commonly used techniques in for solving optimization problems.

§1.3 Research Question

How do the global-best harmony search algorithm and the binary harmonic multi-scale algorithms perform when implemented for solving KP01?

§2. Theory

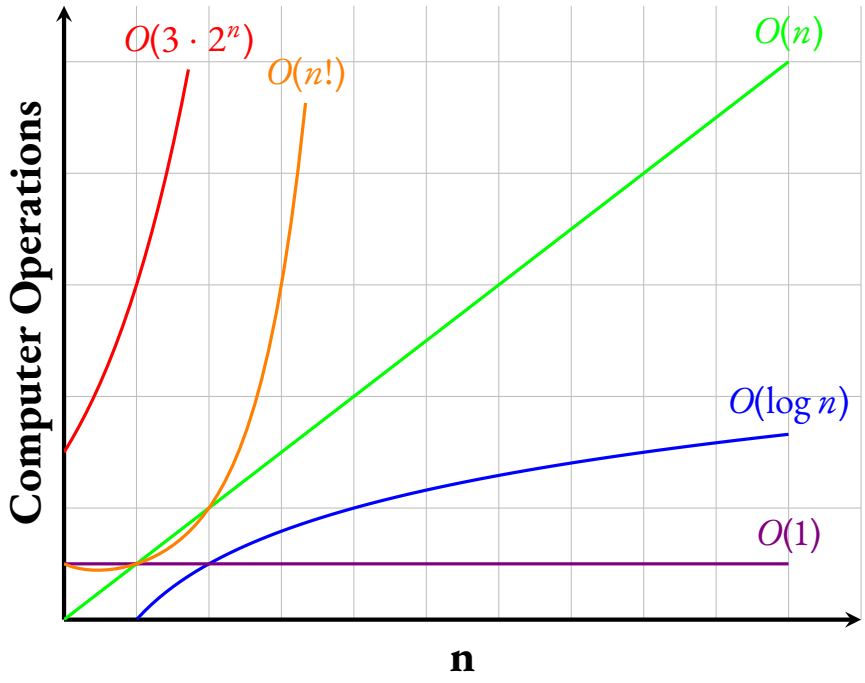
§2.1 Computational Complexity

In computer science, computational complexity is the measure of how expensive it is to the run an algorithm; the amount of resources required to run the algorithm. The 2 resources are time and memory, notated time complexity and space complexity. As the memory complexity is dependent on the time complexity, the time complexity is the limiting factor for the efficiency of an algorithm and is usually the one focused on. Computable problems can be factorial, exponential, polynomial, logarithmic, etc.

Big O notation

The time complexity of an algorithm is a function of the size of the input of that algorithm. For example, if n numbers $a_1, a_2, a_3, ..., a_n$ are given, and an algorithm checks the existence of an certain element in those numbers, by checking all n elements, then the time complexity is some constant times n - the size of the input. Computer scientists would denote this using Big O notation as $O(n)$. Big O notation is a system developed by mathematicians and computer scientists to describe an function/algorithm's asymptotic limiting factor.

Computer scientists usually categories the time complexity of an algorithm into polynomial time complexities and non polynomial time complexities. This is because non-polynomial functions like factorial and exponential functions tend to grow quicker than polynomial functions and thus are usually considered nonviable for large data.



§2.2 Solving the 0-1 Knapsack Problem

There are 2 main approaches to solving KP01: An exact solution using deterministic algorithms, and probabilistic approaches involving heuristic algorithms. A small-scale KP01 can be solved with deterministic approaches, but for high-scale situations it is not realistic to get optimal solutions with exact approaches **??** as the KP01 problem is NP-complete **??**.

Deterministic algorithms

Dynamic programming is a general technique for solving optimization problems. If a problem has optimal substructure and over-lapping subproblems, then dynamic programming is applicable. In computer science a problem has *optimal substructure* if an optimal solution for a problems can be constructed from optimal solutions of its sub-problems, and *over-lapping subproblems* is when a problem can be decomposed into sub-problems which are reused. Dynamic programming breaks down a complicate problem into smaller sub-problems in a recursive manner, while also using some memory to save the solutions of the sub-problems (either in tabular form or memoized form). This way, when we need to get a solution for a sub-problem again, we can just use our previously calculated value.

KP01 is solved using dynamic programming using a table based computation:

Algorithm 1 Pseudocode for solving 0-1 Knapsack with Dynamic Programming

```
for i = 0 to noItems do                                   // If no items, then profit = 0
    Table[i][0] ← 0
end for
for k = 0 to maxCapacity do                               // If no items, then profit = 0
    Table[0][k] ← 0
end for
for i = 0 to noItems do
    for k = 0 to maxCapacity do
        Table[i][k] ← Table[i − 1][k]
        if k − weights[i] >= 0 then
            Table[i][k] ← max(Table[i − 1][k − weights[i]] + profits[i], Table[i][k])
        end if
    end for
end for
Print Table[noItems][maxCapacity]
```

This implementation of dynamic programming method has a time complexity of $O(N \cdot W)$, where N is the number of items, and W is max capacity. The dynamic programming algorithm does not end until the entire table is built. This proves to be inefficient very quickly as the maximum capacity of the knapsack increases. So with the increase in the scale, the feasibility of dynamic programming decreases.

Heuristic algorithms

When deterministic algorithms prove to be too slow for practical applications, heuristic algorithms are chosen for their (usually) near optimal outputs and their speed. Heuristic algorithms should not be confused with approximation algorithms. Approximation algorithms guarantee a maximum margin of error; a constant factor off of the optimal. On the other hand, a heuristic algorithm does not guarantee anything, so it can perform better or worse than an approximation algorithm.

Discrete Global-Best Harmony Search Algorithm

Binary Multi-Scale Quantum Harmonic Oscillator Algorithm