LINEAR & QUADRATIC KNAPSACK OPTIMISATION PROBLEM

Supervisor: Prof. Montaz Ali

Alex Alochukwu, Ben Levin, Krupa Prag, Micheal Olusanya, Shane Josias, Sakirudeen Abdulsalaam, Vincent Langat

January 13, 2018

GROUP 4: Graduate Modelling Camp - MISG 2018

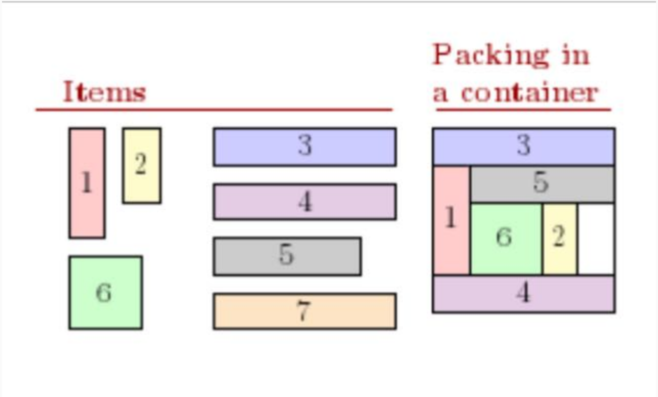
Introduction

- The Knapsack Problem is considered to be a **combinatorial optimization problem**.
- The **best selection and configuration** of a collection of objects adhering to some objective function defines combinatorial optimization problems.

Problem Description: Knapsack Problem

To determine the number of items to include in a collection such that the total weight is less than or equal to a given limit and the total value is maximised.

Problem Description: Knapsack Problem



Mathematical Formulation:Linear Knapsack problem

Given n-tuples of positive numbers $(v_1, v_2, ..., v_n)$, $(w_1, w_2, ..., w_n)$ and $W > 0$. The aim is to determine the subset S of items each with values,v_i and w_i that

$$\text{Maximize} \quad \sum_{i=1}^{n} v_i x_i, \quad x_i \in \{0, 1\},\ x_i \textit{ is the decision variable} \tag{1}$$

$$\text{Subject to:} \quad \sum_{i=1}^{n} w_i x_i < W, \tag{2}$$

$$\text{where} \quad W < \sum_{i=1}^{n} w_i.$$

Quadratic Knapsack Problem

- Extension of the linear Knapsack problem.
- Additional term in the objective function that describes extra profit gained from choosing a particular combination of items.

Mathematical Formulation

$$\text{Maximize} \quad \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j \tag{3}$$

$$\text{Subject to:} \quad \sum_{j=1}^{n} w_j x_j < W, \quad j = \{1, 2, ..., n\}, \tag{4}$$

$$\text{where} \quad x \in \{0, 1\}, \quad \text{Max} \quad w_j \leq W < \sum_{j=1}^{n} w_j.$$

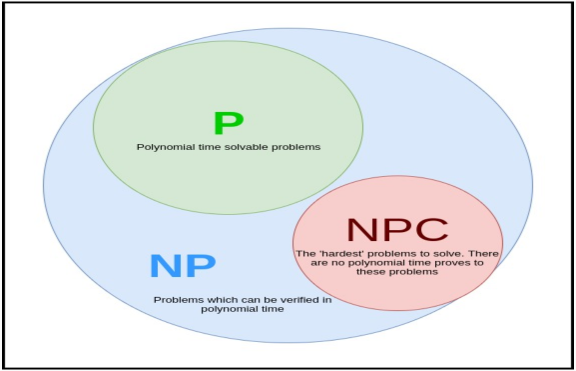
Application

A Knapsack model serves as an abstract model with broad spectrum applications such as:

- Resource allocation problems
- Portfolio optimization
- Cargo-loading problems
- Cutting stock problems

Complexity Theory

- The **abstract measurement** of the rate of growth in the required resources as the **input** *n* **increases**, is how we distinguish among the complexity classes.



Proposed Solution Schemes

The following solution schemes were proposed for solving the Linear and Quadratic Knapsack Problem:

- Greedy Algorithm
- Polynomial Time Approximation Scheme
- Exact Method (Branch and Bound Algorithm)
- Dynamic Programming (Bottom - up)

Greedy Algorithm

1. Identify the available items with their weights and values and take note of the maximum capacity of the bag.
2. Use of a score or efficiency function, i.e. the profit to weight ratio:

$$\frac{v_i}{w_i}.$$

3. Sort the items non-increasingly according to the efficiency function.
4. Add into knapsack the items with the highest score, taking note of their accumulative weights until no item can be added.
5. Return the set of items that satisfies the weight limit and yields maximum profit.

Greedy algorithm for QKP

1. Sample k items from the set of n items.
2. Obtain a set of all pairs from the k items.
3. Sort the items non-increasingly according to the efficiency function

$$S = \frac{d_{ij}}{w_i + w_j}.$$

4. Add into knapsack the pair of items with the highest score, ensuring that the accumulated weight does not exceed the maximum capacity.
5. Repeat steps 1 through 4 until pairs can no longer be added.
6. Fill remaining capacity with singleton items, using the previous greedy approach.

Polynomial Time Approximation Algorithm (PTAS)

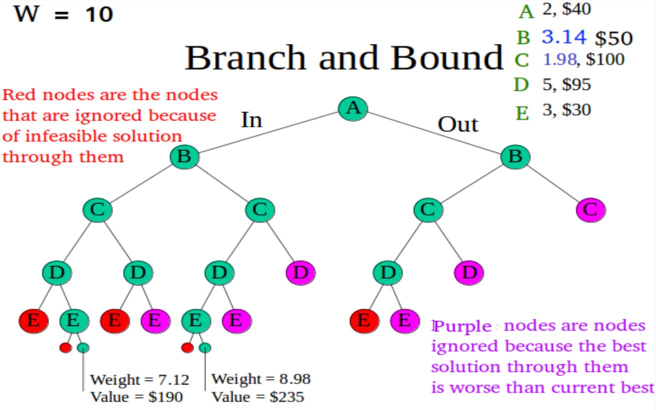
1. Consider all sets of up to at most k items

$$\mathcal{F} = \{F \subset \{1, 2, ..., n\} : |F| \leq k, w(F) < W\}$$

2. For all F in $\mathcal{F}$
   - Pack F into the knapsack
   - Greedily fill the remaining capacity
   - End
3. Return highest valued item combination set

Branch and Bound Method

Branch and Bound performs systematic enumeration of candidate solutions by means of state search space.



Dynamic Programming (DP)

- DP: What is the idea?
- Pros?
- Cons?

Dynamic Programming: Bottom-up

1. Construct $V \in \mathbb{R}^{n \times W}$
   n = Total number of objects to be packed
   W = maximum weight capacity.
   For $1 \leq i \leq n$, and $0 \leq w \leq W$, $V(i, w)$ stores the maximum value of variables $\{1, 2, \ldots, i\}$ of size at most w.
2. $V(n, W)$ is the optimal value of the problem.
3. Recursion
   The process is as follows:
   Initialization:
   $V(0, w) = 0 \forall w \in [0, W]$ (no item); $V(i, w) = -\infty$ if $w < 0$
   Recursive step:
   $V(i, w) = \max(V(i-1, w), v_i + V(i-1, w - w_i))$ for
   $1 \leq i \leq n, 0 \leq w \leq W$.
   $v_i \in \bar{V}$ is the set of values of the objects to be packed while $w_i \in \bar{W}$ is their corresponding weights.

Dynamic Programming- Bottom - Up Approach

Let $W = 10$ and

i	1	2	3	4
v_i	10	40	30	50
w_i	5	4	6	3

w	0	1	2	3	4	5	6	7	8	9	10
i = 0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	50
3	0	0	0	0	40	40	40	40	40	50	70
4	0	0	0	50	50	50	50	90	90	90	90

The optimal value is $V(4, 10) = 90$. The items that give the maximum value are 2 and 4.

Results and Discussions

Table: Algorithm Optimality

	Greedy	PTAS	Dynamic Programming	BB
N = 30	261	261	261	261
N = 50	480	481	481	481
N = 100	891	891	891	891

QKP optimality

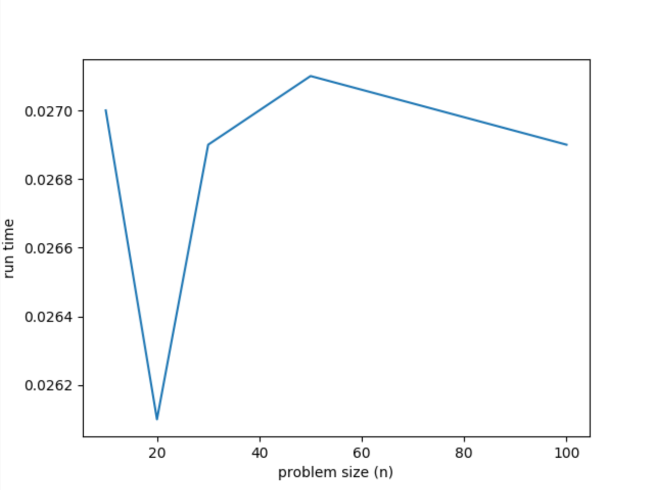
Table: QKP optimality

	$K = 0.5 * N$	$K = N$
N = 10	195	179.5
N = 20	735.5	853.5
N = 30	1739.5	2572.5
N = 50	4755	7528
N = 100	18551.5	16855

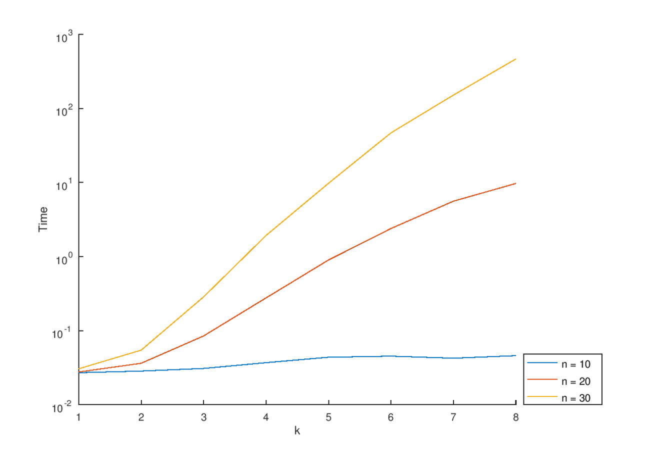
Complexity Analysis of the Algorithms

1. Greedy Algorithm
   - With sorting: $\mathcal{O}(nlogn)$
   - Without sorting: $\mathcal{O}(n^2)$
2. Polynomial Time Approximation Scheme
   - $\mathcal{O}(kn^{k+1})$
3. Dynamic Programming
   - $\mathcal{O}(nW)$

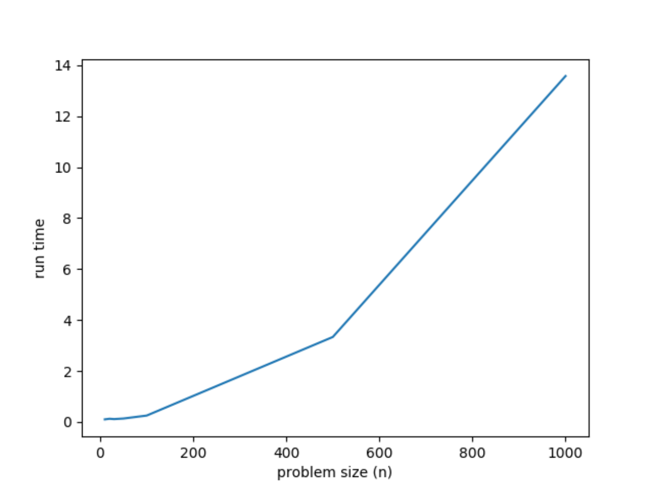
Greedy Runtime



PTAS Runtime



DP Runtime



Conclusion

1. Combinatorial problems are hard to solve.
2. Many applications in industry.
3. Interesting research questions.
4. Better data.

References

K. Lai.
The Knapsack Problem and Fully Polynomial Time Approximation Schemes (FPTAS).
18.434: Seminar in Theoretical Computer Science, 2006.

M. Ali.
Discrete Optimisation .
Lecture notes.