The 0–1 Knapsack Problem
An Introductory Survey

Michail G. Lagoudakis
The Center for Advanced Computer Studies
University of Southwestern Louisiana
P.O. Box 44330, Lafayette, LA 70504
mgl4822@usl.edu

Abstract

The 0–1 Knapsack problem has been studied extensively during the past four decades. The reason is that it appears in many real domains with practical importance. Although its NP-completeness, many algorithms have been proposed that exhibit impressive behavior in the average case. This paper introduces the problem and presents a proof that it belongs to the NP-complete class, as well as a list of directly related problems. An overview of the previous research and the most known algorithms for solving it are presented. The paper concludes with a reference to the practical situations where the problem arises.

1 Introduction

Many people enjoy problem solving and a large collection of puzzlements can be found in the literature. However, most of them would have certainly been surprised when a seemingly simple problem at the beginning turns out to be a very difficult problem in the end.

Consider the following problem:

A hitch-hiker wants to fill up his knapsack, selecting among various objects. Each object has a particular weight and obtains a particular profit. The knapsack can be filled up to a given maximum weight. How can he choose objects to fill the knapsack maximizing the obtained profit?

A naive approach to solve the problem could be the following:

Select each time the object which has the highest profit and fits in the knapsack, until you cannot choose any more objects.

Unfortunately, this algorithm gives a good (or bad) solution which, in general, is not optimal. This is the case also if you try other rules, such as *"choose the object with the highest ratio profit/weight"*. As a result, you may not do anything else, other than testing all the possible subsets of chosen objects. Note that for n objects there are 2^n such subsets. For n=10, 2^n=1024, don't tell about n=20 or more.

The simple problem above is in fact an informal version of an important and famous problem called The O-1[1] Knapsack Problem. This paper studies the problem from the point of view of theoretical computer science. We assume that the reader is familiar with the basics of the theory of NP-completeness and the design and analysis of algorithms.

2 The problem

The 0–1 Knapsack Problem belongs to a large class of problems known as Combinatorial Optimization Problems. In such problems, we try to "maximize" (or "minimize") some "quantity",

[1] This is to emphasize that we cannot choose a fraction of an object or choose it more than one times.

while satisfying some constraints. For example, the Knapsack problem is to maximize the obtained profit without exceeding the knapsack capacity. In fact, it is a very special case of the well-known Integer Linear Programming Problem.

Let us formulate the problem in a mathematical way. We number the objects from 1 to n and define, for all i, 1≤i≤n, the non-negative[2] numbers:
w_i = the weight of object i, and
p_i = the profit of object i.

Also, let W be the capacity of the knapsack, i.e. the maximum weight it can carry. Finally, we introduce a vector X of binary variables x_i (i=1,...,n) having the meaning:

$$x_i = \begin{cases} 1 & \text{, if the object i is selected} \\ 0 & \text{,otherwise} \end{cases}$$

Then the problem is stated as follows: *Find a binary vector X that maximizes the objective function (profit)*

$$\sum_{i=1}^{n} p_i x_i$$

while satisfying the constraint

$$\sum_{i=1}^{n} w_i x_i \leq W$$

Using a set theory notation the problem is: *Given a finite set of objects U, a weight* $w(u) \in Z^+$ *and a profit* $p(u) \in Z^+$ *for each u∈U, along with a knapsack capacity W, find a subset U′ of U such that:*

$$\sum_{u \in U'} w(u) \leq W \text{ and } \sum_{u \in U'} p(u) = max$$

The form given above is the *optimization form* of the problem. In the context of the theoretical study it is reasonable to consider the *decision* (or *recognition*) *form*. That introduces one more parameter P (the desired profit) and asks whether or not there exists a "solution" with profit no less than P. The decision form is given below:

PROBLEM: **0–1 KNAPSACK**
INSTANCE: A finite set of objects U, a weight $w(u) \in Z^+$, a profit $p(u) \in Z^+$ for each u∈U and positive integers W (knapsack capacity) and P (desired profit).
QUESTION: Is there a subset U′ of U such that:

$$\sum_{u \in U'} w(u) \leq W \text{ and } \sum_{u \in U'} p(u) \geq P \text{ ?}$$

The reason for this restriction is that it is desirable to use the theory of NP-completeness (applicable only on decision problems) to derive results for the original problem. The point is that if the objective function can be easily evaluated, the decision problem is no harder than the corresponding optimization problem. So, the implications of the theory can be extended: If the decision problem is proved to be NP-complete, the corresponding optimization problem is NP-hard [GaJo79].

There is an other problem that appears in the literature sometimes with the same name: 0–1 KNAPSACK or SUBSET-SUM. In fact, it is a special case of the problem already described, and asks whether the full capacity of a knapsack can achieved, filling it by choosing among objects with known weights. We call it 0–1 KNAPSACK-FILL to avoid any confusion with the original.

PROBLEM: **0–1 KNAPSACK-FILL (SUBSET-SUM)**
INSTANCE: A finite set of objects U, a weight $w(u) \in Z^+$ for each u∈U and a positive integer W (knapsack capacity).
QUESTION: Is there a subset U′ of U such that:

$$\sum_{u \in U'} w(u) = W \text{ ?}$$

Indeed, the 0–1 KNAPSACK-FILL problem can be derived by the 0–1 KNAPSACK problem by setting w(u)=p(u) for all u∈U, and P=W. One

[2] Instances without this restriction can also be handled by appropriate transformation [MaTo90].

may think that this simpler problem would be easier, however both of them have the same degree of difficulty as it will be revealed in the following section.

3 NP-Completeness

A problem is said to be *tractable* if it can be solved by an algorithm in polynomial time with respect to the "size" of the problem. In the opposite case, it is called *intractable*. Polynomial time algorithms are considered to be "good" algorithms, in the sense that they return a solution in a relatively reasonable amount of time. Moreover, a problem is not thought "well-solved" until a polynomial algorithm is found for it [GaJo79].

In this section we will prove that the two problems described previously are NP-complete. The tractability of the NP-complete problems is an open question. Therefore, they cannot be solved in polynomial time (with respect to the number of the objects), unless P=NP. But this should not be too frustrating, since they can be solved in pseudo-polynomial time as it will be shown in the following section. The proof will be based on the following basic theorem of NP-completeness given without proof.

Theorem 1. Let D1, D2 be decision problems. If D1 belongs to NP, D2 is NP-complete and there is a polynomial time transformation from D2 to D1 (D2 $\leq_p$ D1), then D1 is also NP-complete.

Note that it suffices to prove the NP-completeness of the 0–1 KNAPSACK-FILL problem only. The same follows directly for the 0–1 KNAPSACK problem, since it is a generalization of the 0–1 KNAPSACK-FILL (see [GaJo79] §3.2.1). Two slightly different proofs are given. The difference lies on the selection of the known NP-complete problem to transform into the target problem. The first proof (using EXACT-COVER) is given because it was the first proof historically appeared for the Knapsack problem to be NP-complete. The second (using 3SAT) is more direct, in the sense that 3SAT is, so to speak, the "second" NP-complete problem (SAT being the "first").

PROBLEM: **EXACT-COVER**
INSTANCE: A finite set S and a collection CS of subsets of S.
QUESTION: Does CS contain an exact cover for S, that is a subcollection CS′ $\subseteq$ CS, such that every element of S occurs in exactly one member of CS′?

PROBLEM: **3SAT**
INSTANCE: A finite set of binary variables X={x_1, x_2, ... , x_n} and a collection C={c_1, c_2, ... , c_m} of clauses on X such that $|c_j|$=3 for 1≤j≤m.
QUESTION: Is there a truth assignment for X that satisfies all the clauses in C?

Theorem 2. 0–1 KNAPSACK-FILL (SUBSET-SUM) is NP-complete.

Proof 1. [Karp72] [Sava76] It is obvious that 0–1 KNAPSACK-FILL belongs to NP. A nondeterministic algorithm needs only to guess a subset of objects u_i, add up their weights and check if the summation equals W. This can be done in polynomial time. Therefore, 0–1 KNAPSACK-FILL belongs to NP.

To show that 0–1 KNAPSACK-FILL is NP-hard, we will transform EXACT-COVER to 0–1 KNAPSACK-FILL. Let S={s_1, s_2, ... , s_p} be a set of items and CS={cs_1, cs_2, ... , cs_n} a collection of subsets of S (i.e. $cs_i \subseteq$ S, for i=1,...,n) making up an arbitrary instance of EXACT-COVER. Weights must be assigned to a set of objects U and a knapsack capacity W, so that, CS contains an exact cover of S if and only if there exists a subset U′ of U such that $\sum_{u \in U'} w(u) = W$.

We create |CS|=n objects, so U={u_i, i=1,...,n}. Let β=n+1 and for i=1,...,n and j=1,...,p

$$x_{ij} = \begin{cases} 1, & \text{if } s_j \in cs_i \\ 0, & \text{otherwise} \end{cases}$$

Then the weight for the object u_i is defined as follows

$$w(u_i) = \sum_{j=1}^{p} x_{ij}\beta^{j-1}$$

Finally, the capacity W is defined to be

$$W = \frac{\beta^p - 1}{\beta - 1} = \beta^0 + \beta^1 + ... + \beta^{p-1}$$

The object u_i, i=1,...,n, corresponds to cs_i. The weight of u_i is the summation of some powers of β: β^{j-1} appears in w(u_i) if and only if $s_j \in cs_i$. Therefore, there is a one-to-one correspondence between U and CS and between {β^{j-1}: j=1,...,p} and S. It is obvious that the transformation can be done in polynomial time O(np^2).

Here an example. Consider the following instance of EXACT-COVER:
S={s_1, s_2, s_3, s_4} and CS={cs_1, cs_2, cs_3, cs_4, cs_5}, where cs_1={s_1}, cs_2={s_3, s_4}, cs_3={s_1, s_2, s_4}, cs_4={s_1, s_2}, cs_5={s_2, s_3, s_4}

Then the corresponding instance of 0–1 KNAPSACK-FILL is
U={u_1, u_2, u_3, u_4, u_5}, β=6 and
w(u_1) = 6^0 = 1
w(u_2) = 6^2+6^3 = 252
w(u_3) = 6^0+6^1+6^3 = 223
w(u_4) = 6^0+6^1 = 7
w(u_5) = 6^1+6^2+6^3 = 258

$$W = \frac{6^4 - 1}{6 - 1} = 6^0 + 6^1 + 6^2 + 6^3 = 259$$

It suffices to show that there exists a set U′⊆U such that $\sum_{u \in U'} w(u) = W$ if and only if CS contains an exact cover for S.

Suppose that for some subset U′ of U the summation of the weights equals to W. Then each power of β, β^{j-1} (j=1,...,p), must appear in exactly one of the weights w(u), for u belongs to U′. Since there is a one-to-one correspondence between U and CS and between {β^{j-1}: j=1,...,p} and S, if we take CS′={cs_i: cs_i∈CS and u_i∈U′, i=1,...,n} then CS′ is an exact cover for S.

Conversely, if some subset SC′ of SC is an exact cover for S then each item s_j (j=1,...,p) of S belongs to exactly one cs, for cs belongs to CS′. Since there is a one-to-one correspondence between U and CS and between {β^{j-1}: j=1,...,p} and S, if we take U′={u_i: u_i∈U and cs_i∈CS′, j=1,...,p} then $\sum_{u \in U'} w(u) = W$.

We conclude that EXACT–COVER ≤$_p$ 0–1 KNAPSACK-FILL. By theorem 1 0–1 KNAPSACK-FILL is NP-complete. ■

Proof 2. [MaYo78] It is obvious that 0–1 KNAPSACK-FILL belongs to NP. A nondeterministic algorithm needs only guess a subset of objects u_i, add up their weights and check if the summation equals W. This can be done in polynomial time. Therefore, 0–1 KNAPSACK-FILL belongs to NP.

To show that 0–1 KNAPSACK-FILL is NP-hard, we will transform 3SAT to 0–1 KNAPSACK-FILL. Let X={x_1, x_2, ... , x_n} be a set of variables and C={c_1, c_2, ... , c_m} a set of clauses making up an arbitrary instance of 3SAT. Weights must be assigned to a set of objects U and a knapsack capacity W, so that, C is satisfiable if and only if there exists a subset U′ of U such that $\sum_{u \in U'} w(u) = W$.

The set U of the objects will consist of two kinds of objects:

1. For each variable x_i, 1≤i≤n create two objects u_i and u'_i.
2. For each clause c_j, 1≤j≤m, create two objects $co1_j$ and $co2_j$ (compensating objects).

Now, a weight must be assigned to each of them. The weight w(u) for any u ∈ U will be an (n+m) digit decimal[3] number. The ith digit, corresponds

[3] Actually it could be a base-4 number but let us use decimal for simplicity.

to the variable x_i, while the (n+j)th digit corresponds to the clause c_j. For the objects u_i and u'_i, $1 \le i \le n$, the rightmost n digits are used to identify the corresponding variable x_i, setting the ith digit equal to 1 and the rest (n-1) digits equal to 0. The lefmost m digits are used to identify the clauses c_j, $1 \le j \le m$, where x_i (for u_i) or $\neg x_i$ (for u'_i) appears, setting the (n+j)th digit equal to 1 and the rest equal to 0.

For the compensating objects $co1_j$ and $co2_j$, $1 \le j \le m$, all the rightmost n digits are 0 and the leftmost m digits are used to identify the corresponding clause c_j, setting the (n+j)th digit equal to 1 and the rest (m-1) digits equal to 0. Note that the weights of $co1_j$ and $co2_j$ are identical.

Finally, the capacity W of the knapsack will be an (n+m) digit number with the n rightmost digits equal to 1 and the m rightmost digits equal to 3.

Let us summarize the construction of the 0–1 KNAPSACK-FILL instance:

- $U=\{u_i, u'_i: 1 \le i \le n\} \cup \{co1_j, co2_j: 1 \le j \le m\}$
- For $1 \le i \le n$, $w(u_i)$= (n+m) digit number where:
  - the ith digit is 1
  - the (n+j)th digit, $1 \le j \le m$, is 1 iff x_i appears in c_j
  - all the remaining digits are 0
- For $1 \le i \le n$, $w(u'_i)$= (n+m) digit number where:
  - the ith digit is 1
  - the (n+j)th digit, $1 \le j \le m$, is 1 iff $\neg x_i$ appears in c_j
  - all the remaining digits are 0
- For $1 \le j \le m$, $w(co1_j)=w(co2_j)$= (n+m) digit number where:
  - the (n+j)th digit is 1
  - all the remaining digits are 0
- W= (n+m) digit number where:
  - the n rightmost digits are 1
  - the m leftmost digits are 3

Note that |U|=2(n+m) and each weight (and the capacity) has exactly n+m digits, so the instance can be represented by a table of [2(n+m)+1](n+m) entries. Each entry can be calculated in constant time. Therefore, the transformation from 3SAT to 0–1 KNAPSACK-FILL can be done in polynomial time.

An example is provided for clarification purposes. Consider the following instance of 3SAT: $X=\{x_1, x_2, x_3, x_4, x_5\}$ and $C=\{c_1, c_2, c_3, c_4\}$, where $c_1=\{x_1, \neg x_2, x_4\}$, $c_2=\{x_2, \neg x_3, \neg x_5\}$, $c_3=\{x_3, x_4, x_5\}$, $c_4=\{\neg x_1, x_2, \neg x_5\}$

The corresponding instance of 0–1 KNAPSACK-FILL is the following:
$U=\{u_1, u'_1, u_2, u'_2, u_3, u'_3, u_4, u'_4, u_5, u'_5, co1_1, co2_1, co1_2, co2_2, co1_3, co2_3, co1_4, co2_4\}$

The weight for all objects are shown in the table below. The capacity W is 333311111.

	9	8	7	6	5	4	3	2	1
u_1	0	0	0	1	0	0	0	0	1
u'_1	1	0	0	0	0	0	0	0	1
u_2	1	0	1	0	0	0	0	1	0
u'_2	0	0	0	1	0	0	0	1	0
u_3	0	1	0	0	0	0	1	0	0
u'_3	0	0	1	0	0	0	1	0	0
u_4	0	1	0	1	0	1	0	0	0
u'_4	0	0	0	0	0	1	0	0	0
u_5	0	1	0	0	1	0	0	0	0
u'_5	1	0	1	0	1	0	0	0	0
$co1_1$	0	0	0	1	0	0	0	0	0
$co2_1$	0	0	0	1	0	0	0	0	0
$co1_2$	0	0	1	0	0	0	0	0	0
$co2_2$	0	0	1	0	0	0	0	0	0
$co1_3$	0	1	0	0	0	0	0	0	0
$co2_3$	0	1	0	0	0	0	0	0	0
$co1_4$	1	0	0	0	0	0	0	0	0
$co2_4$	1	0	0	0	0	0	0	0	0
W	3	3	3	3	1	1	1	1	1

It suffices to prove that there exists a set $U' \subseteq U$ such that $\sum_{u \in U'} w(u) = W$ if and only if C is satisfiable.

Suppose that for some subset U' of U the summation of the weights equals to W. The rightmost

n digits ensure that only one of u_i, u'_i, 1≤i≤n, belongs to U′. If both were belonging to U′, the ith digit of the summation would be 2, which contradicts the hypothesis. In fact, these objects form a truth assignment v for the corresponding variables:

- If $u_i \in U'$, then $x_i^v=1$.
- If $u'_i \in U'$, then $\neg x_i^v=1$ or $x_i^v=0$.

The m rightmost digits ensure that each clause is satisfied for this assignment . Indeed, there are three cases for the (n+j)th digit, 1≤j≤m:

1. None of $co1_j$, $co2_j$ belongs to U′. Then $\sum_{u \in U', u=u_i or u=u'_i} w(u) = 3$. So, all the literals in c_j are 1 and, thus, c_j is satisfied.
2. One of $co1_j$, $co2_j$ belongs to U′. Then $\sum_{u \in U', u=u_i or u=u'_i} w(u) = 2$. So, two of the literals in c_j are 1 and, thus, c_j is satisfied.
3. Both $co1_j$, $co2_j$ belongs to U′. Then $\sum_{u \in U', u=u_i or u=u'_i} w(u) = 1$. So, at least one of the literals in c_j is 1 and, thus, c_j is satisfied.

In each case, all c_j are satisfiable and, thus, C is satisfiable.

Conversely, if C is satisfiable then there exists an assignment v, such that at least one of the literals in each clause is 1. For each i, 1≤i≤n, if $x_i^v=1$ then select u_i, else ($x_i^v=0$) select u'_i. This ensures that if the weights of these objects are summed up, the n rightmost digits will be all 1 and the m leftmost digits will be either 1, 2 or 3 (since each clause is satisfied by at least one literal). For 1≤j≤m:

1. If the (n+j)th digit is 1, then select both $co1_j$, $co2_j$.
2. If the (n+j)th digit is 2, then select one of $co1_j$, $co2_j$.
3. If the (n+j)th digit is 3, then select nothing.

This ensures that if all the selected objects are summed up, the n rightmost digits will be all 1 and the m leftmost digits will be all 3, that is, the summation will be equal to W. So, there exists a subset U′ of U, such that $\sum_{u \in U'} w(u) = W$.

Finally, it is concluded that 3SAT $\leq_p$ 0–1 KNAPSACK-FILL. By theorem 1, 0–1 KNAPSACK-FILL is NP-complete. ■

The above discussion results to the following.

Theorem 3. 0–1 KNAPSACK is NP-complete.

Similar proofs using other NP-complete problems can be found in [Papa94], [MaTo90], [Stin87] and [GaJo79].

4 Related problems

In this section a set of related problems is presented. Usually all of the problems are referred under the term Knapsack problems. The decision form is preferred, because it is the common form for theoretical study. The optimization form can be derived easily (see also [MaTo90]).

A generalization of the original 0–1 KNAPSACK problem arises under the assumption that b_i copies of the object u_i, i=1,...n, are available. This is known as the

PROBLEM: **BOUNDED KNAPSACK**
INSTANCE: A finite set of objects U={u_1, u_2, ... ,u_n}, a number of copies $b_i \in Z^+$, a weight $w_i \in Z^+$, a profit $p_i \in Z^+$ for each $u_i \in U$, i=1,...,n, and positive integers W (knapsack capacity) and P (desired profit).
QUESTION: Is there any integer vector X=(x_1, x_2, ... , x_n), such that $0 \leq x_i \leq b_i$ for i=1,...,n and

$$\sum_{i=1}^{n} w_i x_i \leq W \text{ and } \sum_{i=1}^{n} p_i x_i \geq P \text{ ?}$$

For $b_i=+\infty$, i=1,...n, it is called **UNBOUNDED KNAPSACK** problem. A particular case of the BOUNDED KNAPSACK problem, arising when $p_i=1$ and the capacity constraint becomes equality, is called CHANGE-MAKING

problem. The name is related to the situation when a cashier has to assemble a change W giving the maximum (or minimum) number of coins.

PROBLEM: **CHANGE-MAKING**
INSTANCE: A finite set of objects U={u_1, u_2, ... ,u_n}, a number of copies $b_i \in Z^+$, a weight $w_i \in Z^+$, for each $u_i \in U$, i=1,...,n, and positive integers W and P.
QUESTION: Is there any integer vector X=(x_1, x_2, ... , x_n), such that $0 \le x_i \le b_i$ for i=1,...,n and

$$\sum_{i=1}^{n} w_i x_i = W \text{ and } \sum_{i=1}^{n} x_i \ge P \text{ ?}$$

Another generalization of the 0–1 KNAPSACK problem arises when the set of objects U is partitioned into subsets U_1, U_2, ... ,U_k and it is required that exactly one item per subset is selected. This is known as

PROBLEM: **MULTIPLE-CHOICE 0–1 KNAPSACK**
INSTANCE: A finite set of objects U={u_1, u_2, ... ,u_n}, a weight $w_i \in Z^+$, a profit $p_i \in Z^+$ for each $u_i \in U$, i=1,...,n, and positive integers W (knapsack capacity) and P (desired profit). Also a partition U_1, U_2, ... ,U_k of U.
QUESTION: Is there any binary vector X=(x_1, x_2, ... , x_n), such that $\sum_{u_i \in U_j} x_i = 1$ for j=1,...,k and

$$\sum_{i=1}^{n} w_i x_i \le W \text{ and } \sum_{i=1}^{n} p_i x_i \ge P \text{ ?}$$

All the problems introduced so far have been proved to be NP-complete [MaTo90]. Therefore, the corresponding optimization problems are NP-hard. However, all of them can be solved in pseudo-polynomial time by dynamic programming.

If more than one knapsacks, namely k_1, k_2, ... ,k_m with capacities W_1, W_2, ... ,W_m respectively are allowed, the following problem is obtained.

PROBLEM: **0–1 MULTIPLE KNAPSACK**
INSTANCE: A finite set of objects U={u_1, u_2, ... ,u_n}, a weight $w_i \in Z^+$ and a profit $p_i \in Z^+$ for each $u_i \in U$, i=1,...,n. Also, a finite set of knapsacks K={k_1, k_2, ... ,k_m} and a capacity $W_j \in Z^+$ for each $k_j \in K$, j=1,...,m. Finally, a positive integer P (desired profit).
QUESTION: Is there any binary 2–dimensional array X=(x_{ij},), such that $\sum_{j=1}^{m} x_{ij} \le 1$ for i=1,...,n,

$$\sum_{i=1}^{n} w_i x_{ij} \le W_j, \ j = 1, \ldots, m$$

$$\text{and} \sum_{j=1}^{m} \sum_{i=1}^{n} p_i x_{ij} \ge P \text{ ?}$$

Note that the binary variable x_{ij} is 1 if the object u_i has been selected for the knapsack k_j. Note, also, that each object can be chosen only one time and for only one knapsack.

Consider the case where additionally the profit and the weight of each item vary according to the knapsack for which it has been selected. By defining p_{ij} and w_{ij} to be the profit and the weight respectively of the object u_i if it has been selected for the knapsack k_j, the following problem is derived.

PROBLEM: **GENERALIZED ASSIGNMENT**
INSTANCE: A finite set of objects U={u_1, u_2, ... ,u_n}, weights $w_{ij} \in Z^+$ and profits $p_{ij} \in Z^+$ for each $u_i \in U$, i=1,...,n, j=1,...,m. Also, a finite set of knapsacks K={k_1, k_2, ... ,k_m} and a capacity $W_j \in Z^+$ for each $k_j \in K$. Finally, a positive integer P (desired profit).
QUESTION: Is there any binary 2–dimensional

array X=(x_{ij},), such that $\sum_{j=1}^{m} x_{ij} \leq 1$ for i=1,...,n,

$$\sum_{i=1}^{n} w_{ij} x_{ij} \leq W_j, \ j = 1, \ldots, m$$

$$\text{and} \sum_{j=1}^{m} \sum_{i=1}^{n} p_{ij} x_{ij} \geq P \ ?$$

The name of the problem is due to the fact that it can be viewed as the problem of assigning, totally or partially, n jobs to m machines. Each job i requires an amount of resource w_{ij} and obtains a profit p_{ij} when assigned to resource j. The object is to maximize the overall profit without exceeding the availability W_j of the resources.

The last two problems have been proved to be NP-complete *in the strong sense* [MaTo90], so, the corresponding optimization problems are NP-hard in the strong sense. This means that neither polynomial nor pseudo-polynomial algorithm exists for these problems, unless P=NP.

5 Algorithms

5.1 The chronicle

Computer scientists studied the Knapsack problem for the first time in the late fifties. Since then a large number of algorithms and techniques have been proposed for exact or approximate solutions: relaxation, branch and bound, dynamic programming, approximate solutions, parallel algorithms, etc. The table below gives a small chronicle of this research. Although none of these algorithms are pure polynomial, some of them have an impressive record of running quickly in practice. That makes some researchers consider the Knapsack problem as a well solved problem [GaJo79]. An excellent reference on algorithms for Knapsack problems is [MaTo90]. The descriptions presented here are based on [Stin87].

1950's	- First dynamic programming algorithm [Bell57] - Upper bound on the optimal solution [Dant57]
1960's	- Dynamic programming algorithm improvement [GiGo6x] - First branch and bound algorithm [Kole67]
1970's	- Branch and bound algorithm revisited [HoSa74] - First reduction procedure for the number of variables [InKo73] - First polynomial time approximation scheme [Sahn75] - Fully polynomial time approximation scheme [IbKi75] - Upper bound domainating the value of continuous relaxation [MaTo77]
1980's	- Results on the solution of very large size problems where sorting of the variables takes most of the time - Core problem: Sorting only a small subset of the variables [BaZe80]
1990's	- Parallel algorithm [LoSm92] - Neural Networks approach [OhPS93] - AI and learning techniques approach [Ko_I93] - Solving the problem by sampling [PeHA94]

Table 1.1 Historical overview of the research on the problem. (Source: [MaTo90] except "1990's")

5.2 The continuous relaxation

If the restriction for the variables x_i, i=1,...,n, to be 0 or 1 is ommited and any real number between [0,1] is allowed (i.e. any fraction of an object can be chosen), then the Rational Knapsack Problem is obtained. It is the first and the most natural relaxation of the 0–1 Knapsack Problem, originally established by Dantzig in 1957.

The Rational Knapsack problem can be solved optimally by the following *greedy algorithm*:

1. Order and rename the objects so that: $\frac{p_i}{w_i} \geq \frac{p_{i+1}}{w_{i+1}}$ for i=1,...,n-1.
2. Calculate the critical item u_s:

$$s = \min \left\{ i : \sum_{k=1}^{i} w_k > W \right\}$$

3. Calculate the optimal solution X′:
   x'_i=1, for i=1,...,s-1
   x'_i=0, for i=s+1,...,n
   $x'_s = \left(W - \sum_{k=1}^{s-1} w_k \right) / w_s$

The optimal solution X′ of the continuous relaxation gives an upper bound for the optimal solution of the 0–1 Knapsack problem. Moreover, it provides the idea for the following *greedy approximation algorithm* for the 0–1 Knapsack problem (the heuristic given in the introduction):

1. Order and rename the objects so that: $\frac{p_i}{w_i} \geq \frac{p_{i+1}}{w_{i+1}}$ for i=1,...,n-1.
2. CW=0; the current weight
3. For i=1,...n

   a. If $CW+w_i \leq W$
   then $CW \leftarrow CW+w_i$; $x_i=1$
   else $x_i=0$

The time complexity for both of them is polynomial O(n).

5.3 Branch and Bound

A naive approach to solve the 0–1 Knapsack problem is to consider in turn all the 2^n possible solutions (vectors) X, calculating the profit each time and keeping track of the highest profit found and the corresponding vector. Since each x_i, i=1,...,n, can be only 0 or 1, this can be done by a *backtracking algorithm* traversing a binary tree in a depth-first fashion. The level i of the tree corresponds to variable x_i. The internal nodes at some level represent partial solutions extended at the next level. For example, x=(0,1,-,-,-) has two children: (0,1,0,-,-) and (0,1,1,-,-). So the leaves represent all the possible solutions (feasible or not). But this *exhaustive algorithm* leads to an exponential complexity $\Theta(n2^n)$, which is not desirable.

The average case can be improved in two ways:

1. Pruning the branches that leads to non-feasible solutions. This can be done by calculating the current weight at each node, i.e. the weights of the objects selected so far in the partial solution. If the current weight at some node is greater than the Knapsack capacity W the subtree under this node is pruned; it leads to non-feasible solutions.
2. Pruning the branches that will not give a better profit than the optimal so far. This can be done by calculating an upper bound for the solutions under each node (bounding function) and comparing it with the optimal profit found so far. If the bound is less (or equal) than the optimal so far, the branch is pruned. The *bounding function* at a node can be calculated by adding the profits of the objects selected so far in the partial solution and the optimal solution obtained from the relaxation of the "remaining" problem using the greedy algorithm.

In fact this approach is not used in a pure depth-first fashion, but in a best-first fashion: Calculate the bounding function on both sons of a node and follow first the most "promising" branch, i.e. the one with the greatest value.

The *branch and bound algorithm* described above has an exponential time complexity $O(n2^n)$ in the worst case, but it is very fast in the average case. More complicated branch and bound schemes have also been proposed [MaTo90].

5.4 Dynamic Programming

The first dynamic programming approach described here is based on the following idea. The value of, say, x_n can be either 0 or 1. If $x_n=0$, then the best profit possibly obtained is whatever is attained from the remaining n-1 objects with knapsack capacity W. If $x_n=1$, then the best profit possibly obtained is the profit p_n (already selected) plus the best profit obtained by the remaining n-1 objects with knapsack capacity $W-w_n$ (in this case it must be $w_n \leq W$). Therefore, the optimal profit will be the maximum of these two "best" profits.

The idea above leads to the following recurrence relation, where P(i,m), i=1,...,n, m=0,...,W, is the best profit obtainable from the objects 1,...,i

with knapsack capacity m:

$$P(i,m) = \begin{cases} P(i-1,m) & , w_i > m \\ max\{^{P(i-1,m)}_{P(i-1,m-w_i)+p_i}\} & , w_i \leq m \end{cases}$$

with initial conditions

$$P(1,m) = \begin{cases} 0 & , w_1 > m \\ p_1 & , w_1 \leq m \end{cases}$$

So, the optimal profit is P(n,W).

A *dynamic programming algorithm* simply has to construct an n×W table and calculate the entries P(i,m) (i=1,...n, m=0,...,W), in a bottom-up fashion. As soon as the optimal profit P(n,W) has been calculated, the optimal solution X can be found by backtracking through the table and assigning 0's and 1's to x_i's according to the selections of the max function.

The algorithm has a time complexity of O(nW). Note that this is not polynomial, since W can be exponentially large to n, e.g. $W=2^n$. Such a complexity is called *pseudo-polynomial*.

An other dynamic programming approach attempts to minimize the knapsack capacity required to achieve a specific profit. For i=1,...,n and p=0,..., $\sum_{k=1}^{i} p_k$, W(i,p) is defined to be the minimum knapsack capacity into which a subset of the objects 1,...,i, obtaining a profit of at least p, can fit. Then W(i,p) satisfies the following recurrence relation:

$$W(i,p) = \begin{cases} W(i-1,p-p_i) + w_i & , \sum_{k=1}^{i-1} p_i < p \\ min\{^{W(i-1,p)}_{W(i-1,p-p_i)+w_i}\} & , \sum_{k=1}^{i-1} p_i \geq p \end{cases}$$

with initial conditions

$$W(1,p) = \begin{cases} w_1 & , p \leq p_1 \\ +\infty & , p > p_1 \end{cases}$$

Then the optimal profit is

$$P = max\{p : W(n,p) \leq W\}$$

A *dynamic programming algorithm* simply has to construct an n× $\sum_{k=1}^{n} p_k$ table and calculate the entries W(i,p) (i=1,...n, p=0,..., $\sum_{k=1}^{i} p_k$), in a bottom-up fashion. As soon as the optimal profit P has been calculated, the optimal solution X can be found by backtracking through the table and assigning 0's and 1's to x_i's according to the selections of the min function. The algorithm has a pseudo-polynomial complexity $O(n^2 maxp)$, where maxp=max{p_i: i=1,...,n}.

5.5 Approximate Algorithms

Occasionally a small loss of accuracy is acceptable for the sake of speed in running time. A near-optimal solution is often as good as an optimal solution, but it can be obtained in significantly less time. This is the reason that several approximation algorithms have been proposed for the Knapsack problem in the literature.

The first approximation algorithm described here is based on the second dynamic programming approach already presented above. Recall that the complexity was $O(n^2 pmax)$. The idea is that, if the profits can be scaled in some manner, so that pmax is reduced, it will lead to a significant improvement in the running time. Given a positive integer k, this scaling can be done by the following substitution:

$$p_i \leftarrow \left\lfloor \frac{p_i}{2^k} \right\rfloor \text{ ,i=1,...,n}$$

In fact, the last k bits of the binary representation of each p_i are truncated. The result is a small loss of accuracy but now the computational complexity has been reduced to $O(n^2 pmax/2^k)$. It can be proved that the relative error

$$re = \frac{P_{opt} - P_{app}}{P_{opt}}$$

where P_{opt} and P_{app} are the profits of the optimal and the approximate solution respectively, is bounded by the quantity

$$\frac{n\left(2^k - 1\right)}{pmax}$$

The integer k can by chosen using the following formula if we desire the relative error to be at most ϵ>0:

$$\frac{n\left(2^k-1\right)}{pmax} \le \epsilon \text{ or } k \le \log_2\left(1+\frac{\epsilon \cdot pmax}{n}\right)$$

and since k must be integer

$$k = \left\lfloor \log_2\left(1+\frac{\epsilon \cdot pmax}{n}\right)\right\rfloor$$

The complexity now becomes $O(n^3/\epsilon)$. For a fixed ϵ>0, we have an *ϵ-approximation algorithm* with complexity $O(n^3)$.

Another approximation algorithm can be derived in the following way. Suppose that the greedy approximation algorithm is used to produce a feasible solution X1 with profit P1 for an instance I. Let, also, X2 with profit P2 be the feasible solution that contains only the object with the highest profit. Comparing P1, P2 we can obtain a feasible solution X^* with profit P^*=max{P1, P2}. It has been proven that if P_{opt} is the profit of the optimal solution, then:

$$P^* \le P_{opt} \le 2P^*$$

Then the relative error is bounded by 0.5 and this is a simple *0.5–approximation algorithm*. Now, by taking

$$k = \left\lfloor \log_2\left(1+\frac{\epsilon \cdot P^*}{n}\right)\right\rfloor$$

for some ϵ and apply the ϵ-approximation algorithm above to the initial instance I, it can be proven that the relative error is at most ϵ and the time complexity becomes $O(nP_{opt}/2^k)$ which is (n^2/ϵ). Therefore, this improvement leads to another *ϵ-approximation algorithm* with better complexity.

6 Applications

The 0-1 Knapsack problem is one of the few problems that have been extensively studied during the last few decades. There are three main reasons [MaTo90]:

a. It is a special case of the very important problem of Integer Linear Programming.
b. It appears very often as a subproblem into other problems.
c. It arises in many practical situations, such as cargo loading, cutting stock, etc.

Suppose we have to invest (totally or partially) an amount of W dollars and we are considering n investments. If p_i is the profit expected from investment i and w_i the amount of dollars it requires, then finding the optimal choice of investments, i.e. the one that maximizes the profit, is equivalent to solving the corresponding knapsack problem. Suppose, also, that a transportation company has to transport a stack of n items. If item i has weight w_i and each truck can carry items with total weight up to W, which is the ideal amount for the trucks to carry as much weight as possible on each routing? The knapsack (truck) problem appears again.

The problem has been used also in cryptography for public key encryption. Other domains where the problem appears are: cargo loading, project selection, budget control, capital budgeting, cutting-stock, network flow, simulated annealing, selection of journals for a library, usage of group theory in integer programming, memory sharing, stochastic signals, etc. A good overview of the early applications is located in [SaKl75].

7 Conclusion

This paper provides an introduction to the Knapsack problem for the unfamiliar reader. By no means it is a complete reference. The following references and the extensive list in [MaTo90] provide pointers to the literature for the demanding reader.

8 Acknowledgments

I would like to thank my professors Dr. William R. Edwards and Dr. Gui-Liang Feng for their useful comments. Also, I would like to thank Mrs Deana Michna for her invaluable help in the revision of the text.

9 References

[BaZe80] Balas, E. & Zemel, E. “An algorithms for large zero-one knapsack problems”, in *Operations Research* **28**, 1980, pp. 1130–1154.

[Bell57] Bellman, R. *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[Dant57] Dantzig, G.B. “Discrete variable extremum problems”, in *Operations Research* **5**, 1957, pp. 266–277.

[GaJo79] Garey, M.R. & Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Comp., San Francisco, 1979.

[GiGo61] Gilmore, P.C. & Gomory, R.E. “A linear programming approach to the cutting stock problem I”, in *Operations Research* **9**, 1961, pp. 849–858.

[GiGo63] Gilmore, P.C. & Gomory, R.E. “A linear programming approach to the cutting stock problem II”, in *Operations Research* **11**, 1963, pp. 863–888.

[GiGo66] Gilmore, P.C. & Gomory, R.E. “The theory and computation of Knapsack functions”, in *Operations Research* **14**, 1966, pp. 1045–1074.

[HoSa74] Horowitz, E. & Sahni, S. “Computing partitions with applications to the knapsack problem”, in *Journal of ACM* **21**, 1974, pp. 277–292.

[IbKi75] Ibarra, O.H. & Kim, C.E. “Fast approximation algorithms for the knapsack and sum of subset problems”, in *Journal of ACM* **22**, 1975, pp. 463–468.

[InKo73] Ingargiola, G.P. & Korsh, J.F. “A reduction algorithm for zero-one single knapsack problems”, in *Management Science* **20**, 1975, pp. 460–463.

[Karp72] Karp, R.M. “Reducibility among combinatorial problems”, in Miller, R.E. & Thatcher, J.W. (eds.) *Complexity of Computer Computations*, Plenum Press, New York, 1972, pp. 85–103.

[Ko_I93] Ko, I. “Using AI techniques and learning to solve multi-level knapsack problems”, *PhD Thesis*, University of Colorado at Boulder, Boulder, CO, 1993.

[Kole67] Kolesar, P.J. “A branch and bound algorithm for the knapsack problem”, in *Management Science* **13**, 723–735.

[LoSm92] Loots, W. & Smith, T.H.C. “A parallel algorithm for the zero-one knapsack problem”, in *Int. J. Parallel Program.* **21**, 5, 1992, pp. 313–348.

[MaYo78] Machtey, M. & Young P. *An Introduction to the General Theory of Algorithms*, Elsevier North-Holland, Inc., New York, 1978, Ch.7.

[MaTo77] Martello, S. & Toth, P. “An upper bound for the zero-one knapsack problem and a branch and bound algorithm”, in *European Journal of Operational Research* **1**, 1977, pp. 169–175.

[MaTo90] Martello, S. & Toth, P. *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, New York, 1990.

[OhPS93] Ohlsson, M., Peterson, C. & Soderberg, B. “Neural Networks for optimization problems with inequality constraints: the knapsack problem”, in *Neural Computation* **5**, 2, 1993, pp. 331–339.

[Papa94] Papadimitriou, C.M. *Computational Complexity*, Addison-Wesley Publishing Company, Inc., Reading, MA, 1994, Ch. 9.

[PeHA94] Penn, M., Hasson, D. and Avriel, M. “Solving the 0–1 prportional Knapsack problem by sampling”, in *J. Optim. Theory Appl.* **80**, 2, 1994, pp. 261–272.

[Sahn75] Sahni, S. “Approximate algorithms for the 0–1 knapsack problem”, in *Journal of ACM* **22**, 1975, pp. 115–124.

[SaKl75] Salkin, H.M. & de Kluyver, C.A. “The knapsack problem: a survey”, in *Naval*

Research Logistics Quarterly **22**, 1975, pp. 127–144.

[Sava76] Savage, J.E. *The complexity of computing*, John Wiley & Sons, Inc., New York, 1976, Ch. 8.

[Stin87] Stinson, R.D. *An Introduction to the Design and Analysis of Algorithms (2nd Ed.)*, Winnipeg, Manitoba, Canada, 1987, Ch. 3,4,6.