Algorithmic Modeling and Complexity **Fall 2003**

Lecture 13: October 14, 2003

Lecturer: J. van Leeuwen *Scribe: B. de Boer*

13.1 Overview

In modeling industrial processes, packing and cutting problems frequently arise. Packing problems are known to be computationally hard. This lecture focused on packing problems and more specifically, on the 1-dimensional packing problem or *knapsack problem*. We prove that (the decision version of) the 1-dimensional packing problem is NP-complete by showing that Vertex Cover $\preccurlyeq_p$1-dimensional packing.

Later on, a pseudo-polynomial time algorithm is explained and used in solving the 1-dimensional packing problem approximately to arbitrary precision and yet in reasonable computation time. Finally, we define the problem a little bit differently, by allowing fractional solutions. We then look at a greedy packing algorithm and show that it works in polynomial time and within a performance ratio of 2.

13.2 The 1-dimensional packing problem

The 1-dimensional packing problem is defined as follows.

Given:

- a set S of objects a_i, where each object has:
  - a size $s(a_i) = s_i$
  - a profit $p(a_i) = p_i$
- and a bound B

determine a subset of the objects that has a total size $\leq B$ and whose total profit is maximized.

We assume thoughout that B and the s_i, p_i are integer and ≥ 0.

In one dimension, you can look at this as trying to pack objects with a certain length or weight into a bin that can fit objects up to a certain total length or weight, respectively. This problem occurs frequently in industrial contexts when a certain amount of material has to be divided over orders. Consider for instance cutting pieces of textile from a role, where the pieces correspond to orders which return a profit. This example also illustrates why cutting and packing often give rise to similar problems.

Of course packing is not limited to just one dimension. 3-Dimensional packing for instance is important in the *container loading* problem. However, in this lecture only 1-dimensional packing is discussed. Another interpretation of this problem is to pack a backpack with objects that have a certain size and a certain value, such that the largest total value is packed. This is known as the $0-1$ *Knapsack Problem*, meaning that there is only one copy of each object and each object can only be packed 0 or 1 time.

13.2.1 Linear Programming model of 1-dimensional packing

We can construct the following Linear Programming model of the 1-dimensional packing problem:

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

subject to:

$$\begin{aligned} &\textstyle\sum_{i=1}^{n} s_i x_i \leq B \\ &x_i \in \{0,1\} \qquad \text{for } i = 1, \ldots, n. \end{aligned}$$

A rough solution to this problem would be to just check all 2^n possible combinations of values for the x_i's.

13.3 NP-completeness of 1-dimensional packing

1-Dimensional packing is a deceptive problem: *no polynomial-time algorithm is currently known for it.* In fact, it belongs to the hardest problems in the class NP.

Theorem 13.1 *1-Dimensional packing is NP-complete.*

Proof: Consider the decision version of 1-dimensional packing: given integers B and C, are there $0-1$ values for the x_i such that $z = \sum_{i=1}^{n} p_i x_i \geq C$ while $\sum_{i=1}^{n} s_i x_i \leq B$ ('is there a feasible packing with total profit $\geq C$'). It is trivial to see that this decision problem is $\in$ NP. To prove NP-completeness, we show that Vertex Cover $\preccurlyeq_p$ 1-dimensional Packing.

Consider an instance of the Vertex Cover problem: given a network $G =< V, E >$ with m edges and a number K, does G have a vertex cover of size $\leq K$? We will build an instance of the 1-dimensional packing problem equivalent to it.

Define constants e_{it} for $i \in V$ and $t \in E$ as follows:

$$e_{it} = \begin{cases} 1 & \text{if node } i \text{ is incident to edge } t, \\ 0 & \text{otherwise.} \end{cases}$$

Choose a large enough radix g. Now we create an instance of 1-dimensional packing with the following objects:

- an object a_i with $size(a_i) = profit(a_i) = g^{m+1} + \sum_{t=1}^{m} e_{it} \cdot g^t$ for every $i \in V$.
- K auxiliary objects of size and profit g^{m+1}.
- 1 extra auxiliary object of size and profit g^t for every edge t.

Let $B = K \cdot g^{m+1} + \sum_{t=1}^{m} 2 \cdot g^t$, and take $C = B$: as profits and sizes are equal, this means that the decision problem asks for a packing of *exactly* size B.

Claim 13.2 *G has a vertex cover of size $\leq K$ if and only if the instance of the 1-dimensional packing problem has a positive solution.*

($\Rightarrow$) Consider a vertex cover of $K' \leq K$ nodes. Adding the objects a_i corresponding to the nodes i in the vertex cover gives a packing of size:

$$K' \cdot g^{m+1} + \sum_{t=1}^{m} [1 \text{ } or \text{ } 2] \cdot g^t.$$

The coefficient of g^t is 1 or 2 depending on whether one or both end points of edge t are covered by the vertex cover. Adding $(K - K')$ objects of size $g^m + 1$ plus at most one object of size g^t for every edge t, gives a packing of exactly size (and profit):

$$B = K \cdot g^{m+1} + \sum_{t=1}^{m} 2 \cdot g^t = C$$

($\Leftarrow$) Suppose a bunch of objects solve the packing instance, i.e. their sizes add up to exactly

$$B = K \cdot g^{m+1} + \sum_{t=1}^{m} 2 \cdot g^t.$$

Notes that when we add object sizes, the coefficient of g^t can be at most 3. If we take $g \geq 4$, then the coefficients in the lower order terms do not carry and we can uniquely 'decipher' B. Thus: take $g = 4$.

Consider the packing. Take away all the auxiliary objects! This leaves us with a packing of size:

$$B' = K' \cdot g^{m+1} + \sum_{t=1}^{m} [1 \text{ } or \text{ } 2] \cdot g^t$$

Thus this can arise only by adding K' objects a_i. The corresponding nodes must cover every edge t at least once and thus form a vertex cover of G of size $K' \leq K$. ■

The given proof has another implication: it even shows the NP-completeness of a very special case of 1-dimensional packing, namely the case when $size(a_i) = profit(a_i)$ for every i. This is known as the *Subset Sum* problem:

> Given a set of objects $S = \{a_1, \ldots, a_n\}$, sizes s_i and an integer B. Determine whether there is a subset of S with a total size that is exactly equal to B.

Corollary 13.3 *The Subset Sum problem is NP-complete.*

13.4 A fast and tuneable approximation scheme

The NP-completeness of the 1-dimensional packing problem suggests that large instances better be solved to approximate optimality, using a heuristic that is polynomial-time computable. We show that this can be done by the technique of *scaling.* The key ingredient for this is an exact solver.

13.4.1 An exact solution

To solve the 1-dimensional packing problem one may try a ‘tabular’ technique that attempts to build an solution bottom-up from the optimal solution of all possible subproblems (‘dynamic programming’). Let T be the two-dimensional table we want to construct. We want the following property:

> $T[i, p]$ is a subset of $\{a_1 ... a_i\}$ of minimum possible total size which has a profit of exactly p.

Thus, T will be a n-by-$(P + 1)$ table, with $P = \sum_{i=1}^{n} p_i$ (the table has a first column corresponding to $p = 0$). If we manage to compute T, then the answer to the 1-dimensional packing problem can be retrieved from the n-th row: *take the largest p such that $T[n, p]$ is defined and size$(T[n, p]) \leq B$.*

Lemma 13.4 *The 1-dimensional packing problem can be solved in $O(n \cdot P)$ time (thus in time proportional to the size of table T).*

Proof: T can be filled efficiently row after row, as follows. The first row by definition consists of the following ‘values’:

$$T[1, p] = \begin{cases} \{a_1\} & \text{if } p = p_1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Suppose we have computed the values in row j. Then the values in row $j + 1$ can be computed from the following inductive formula:

$$T[j + 1, p] = \begin{cases} T[j, p] & \text{if } p < p_{j+1}, \\ T[j, p] & \text{if } p_{j+1} = 0, \\ \text{the smallest in size of } T[j, p] \text{ and } \{a_{j+1}\} & \text{if } p = p_{j+1} > 0, \\ \text{the smallest in size of } T[j, p] \text{ and } T[j, p - p_{j+1}] \cup \{a_{j+1}\} & \text{if } p > p_{j+1}. \end{cases}$$

In computing the entries in row $j + 1$ it seems we may have to repeatedly copy entire sets from the previous row. To avoid this, we change the representation slightly. In each entry

$T[j+1,p]$ we will not store the explicit set-value but only (i) the size of the set in $T[j+1,p]$, (ii) a bit b_1 that says whether the set in this slot equals the one in $T[j,p]$ or not, and (iii) a bit b_2 that says whether the set in this slot equals $\{a_{j+1}\}$ or $T[j,p-p_{j+1}] \cup \{a_{j+1}\}$ (if $b_1 = 1$ then b_2 is formally undefined and can be set to any value as it will not be inspected). For example, if $b_1 = b_2 = 0$, then we know that the set $T[j+1,p]$ consists of $\{a_{j+1}\}$ *and* (the elements of) the set represented in entry $T[j,p-p_{j+1}]$. (Note that the backpointer value $-p_{j+1}$ follows implicitly from the row index.) The actual set can be reconstructed entirely by 'backchaining' from this entry. On the other hand, we do not need the complete set for computing the three items in $T[j+1,p]$: the representations in the entries in row j suffice.

It is easily seen that with the new representation, the entries in T can be computed row after row in $O(1)$ time per entry, thus in $O(n \cdot P)$ time total. The largest p such that $T[n,p]$ is defined and $size(T[n,p]) \leq B$ can be found in $O(P)$ time by going through row n. Retrieving the set that realizes it can be done by backchaining from entry $T[n,p]$, which takes $O(n)$. ∎

13.4.2 Pseudo-polynomiality

The given algorithm for solving the 1-dimensional packing problem is *not* a polynomial-time algorithm! The reason is that $P = \sum_{i=1}^{n} p_i$ is not polynomially bounded in the size of the problem, which has all numbers and especially the p_i represented in binary: thus the p_i are written down in only $\sum_{i=1}^{n} \log p_i$ bits and P is 'exponential' in this measure!

If the numbers in the 1-dimensional packing problem would have been written in *unary* notation, then the given algorithm would have been 'polynomial' in the size of the input. Algorithms that are polynomial-time bounded if problem instances are specified in unary notation, are called *pseudo-polynomial* time algorithms.

Pseudo-polynomial time algorithms generally do poorly when the input contains 'very large numbers'. On the other hand, they can be very useful if there is a low upperbound on the size of the numbers. In our case, the exact solver for 1-dimensional packing can be very useful in applications where P is not too large. Note that the runtime of the algorithm can also be written as $O(n^2 p_{max})$, where p_{max} is the largest profit value of any object in a given instance.

13.4.3 An algorithm scheme using scaling

We can avoid that P is large in instances of 1-dimensional packing by *scaling* the profit values p_i. In doing so we loose some accuracy in the scaled values, and we can only hope for a result that will be 'close to optimal' in the end. We show that the scaling can be tuned so a performance ratio $\leq 1+\epsilon$ results, for any desired $\epsilon > 0$. Before the algorithm is run, we eliminate all objects that do not fit, i.e. we assume that $p_{max} \leq B$ from now on.

```
Algorithm 1dP

specify $\varepsilon > 0$ (we assume that $\varepsilon \leq \frac{1}{2}$ without loss of generality)

let $K$ be a suitable scaling factor (to be determined later)

scale the profits: $p'_i := \lfloor \frac{p_i}{K} \rfloor$
```

run the pseudo-polynomial solver on the instance with the same sizes but profits p'_i.

return the subset of objects S' computed by the solver as answer.

Note that after scaling: $p_i - K \leq K \cdot p'_i \leq p_i$. The choice of K will be given in the proof below.

Theorem 13.5 (Ibarra and Kim, 1975) *Algorithm 1dP computes a feasible solution to the 1-dimensional packing problem within performance ratio $\leq 1+\varepsilon$ and a running time of $O(\frac{n^3}{\varepsilon})$.*

Proof: Let the optimum solution of a given instance have profit OPT, and assume w.l.o.g. that it is realized by packing the objects $1, \ldots, k$. Let algorithm 1dP compute a solution OPT', realized by objects $i_1, \ldots i_t$. Then the profit of the computed solution satisfies:

$$p_{i_1} + \cdots + p_{i_t} = K \cdot (\frac{p_{i_1}}{K} + \cdots + \frac{p_{i_t}}{K}) \geq K \cdot (\lfloor \frac{p_{i_1}}{K} \rfloor + \cdots + \lfloor \frac{p_{i_t}}{K} \rfloor) \geq K \cdot OPT'.$$

On the other hand, the objects $1, \ldots, k$ form a feasible solution to the scaled problem. It means that $OPT' \geq \lfloor \frac{p_1}{K} \rfloor + \cdots + \lfloor \frac{p_k}{K} \rfloor$ and thus

$$K \cdot OPT' \geq K \cdot (\lfloor \frac{p_1}{K} \rfloor + \cdots + \lfloor \frac{p_k}{K} \rfloor) \geq K \cdot (\frac{p_1}{K} - 1 + \cdots + \frac{p_k}{K} - 1) = (p_i + \cdots + p_k) - K \cdot k \geq OPT - K \cdot n.$$

By combining the two estimates it follows that we have a feasible solution with (unscaled) profit $\geq OPT - K \cdot n$.

How close to OPT can we make this? Clearly $OPT \geq p_{max}$, as the object with profit p_{max} by itself is already feasible (by assumption). Choose $K = \frac{\varepsilon \cdot p_{max}}{2n}$. Then

$$OPT - K \cdot n = OPT - \frac{1}{2}\varepsilon \cdot p_{max} \geq (1 - \frac{1}{2}\varepsilon) \cdot OPT \geq \frac{1}{1+\varepsilon} OPT$$

using that $\varepsilon \leq \frac{1}{2}$, and thus the performance ratio of the algorithm is $\leq 1+\varepsilon$.

The running time of algorithm 1dP determined by the running time of the pseudo-polynomial algorithm on the scaled profits. It is in the order of:

$$n \cdot \sum_{i=1}^{n} \lfloor \frac{p_i}{K} \rfloor \leq n \cdot \sum_{i=1}^{n} \frac{p_i}{K} = n \cdot \frac{P}{K} = \frac{2n^2}{\varepsilon} \cdot \frac{P}{p_{max}} \leq \frac{2n^3}{\varepsilon}.$$

where we use that $\frac{P}{p_{max}} \leq n$. ■

Thus, algorithm 1dP can be tuned to any desired to any small ε that is desired, while remaining polynomial in n and $\frac{1}{\varepsilon}$. 1dP is called a *fully polynomial-time approximation scheme.*

Exercise. Consider algorithm 1dP. Show that the choice of K guarantees that, if the objects do not all have profit 0, then the scaled profits will not all be 0 either. (Hint: consider p'_{max}.)

Exercise. Modify algorithm 1dP to a polynomial-time algorithm of similar qualities but with K a power of 2. (Hint: when 1dP would set K to a value < 1, then all profit values must be 'polynomially bounded'.)

By refining the algorithm and dealing with large and small objects separately in a suitable way, an even better algorithm can be obtained.

Theorem 13.6 (Lawler, 1979) *The 1-dimensional packing problem can be solved by an algorithm scheme that achieves a performance ratio $\leq 1 + \varepsilon$ and has a running time of $O(n \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon^4})$.*

13.4.4 Approximation by a greedy approach

In the 1-dimensional packing problem a problem must be either packed fully or not at all. This limits us in 'filling holes'. We first consider the relaxed packing problem in which we allow that *fractions* ('parts') of objects are packed. We then use it in the design of an approximation algorithm for the original 1-dimensional packing problem.

13.4.5 Fractional 1-dimensional packing

Consider the 1-dimensional packing problem and allow that fractions of objects are packed. The problem is called the *knapsack problem* rather than the $0 - 1$ knapsack problem. The profit of a fraction x_i ($0 < x_i \leq 1$) of object a_i will be $x_i p_i$. The LP model becomes:

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

subject to:

$$\begin{aligned} &\textstyle\sum_{i=1}^{n} x_i s_i \leq B \\ &0 \leq x_i \leq 1 \qquad \text{for } i = 1, \ldots, n. \end{aligned}$$

We will show that the fractional 1-dimensional packing problem can be solved in polynomial time in a simple way.

Assume the n objects in a given instance are ordered by decreasing 'profit per size' ratio. This requires only a simple sorting step. Thus assume the objects are listed such that:

$$\frac{p_1}{s_1} \geq \frac{p_2}{s_2} \geq \ldots \geq \frac{p_n}{s_n}.$$

Now, pack *greedily*: pick the objects in the order of the list for as long as possible, thus always packing the available object with the largest $\frac{p}{s}$-ratio first until we reach the size-limit of B. If the last object we picked does not fit anymore in its totality, pack the fraction of the object that fills up the bin.

Theorem 13.7 *The greedy algorithm solves the fractional 1-dimensional packing problem* optimally, *in $O(n \log n)$ time.*

Proof: The algorithm follows the intuition. The argument to prove optimality is the following. Consider any optimal solution to the fractional problem. Suppose there are objects i and j with $i < j$ in it such that object i is not fully packed ($x_i < 1$) but object j is present for at least a non-zero fraction ($x_j > 0$). Consider a 'slice' of size $b > 0$ filled by (a part of) object j with b small enough so we could exchange it for a 'slice' of size $b > 0$ from the unpacked part of object i. Note that

$$\frac{b}{s_j} \cdot p_j \leq \frac{b}{s_i} \cdot p_i$$

thus exchanging the part of j by the (still unpacked) part of the same size of i will 'increase' the overall profit. Thus any optimal solution can be put into a normal form in which all objects of largest $\frac{p}{s}$-value that fit are fully packed in the bin. This is exactly how the greedy algorithm does the packing. ■

Interestingly, the greedy algorithm leads to an optimal solution in which *at most one* object will be packed 'fractionally', i.e. with at most one value x_i with $0 < x_i < 1$.

13.4.6 Greedy algorithm for 1-dimensional packing

Let's return to the ordinary (non-fractional) 1-dimensional packing problem. An obvious idea is to (i) run the greedy algorithm but (ii) run it only *up* to the point that the fractional object would be included. This gives a feasible solution, but how good is it?

Assume the objects are ordered again by decreasing $\frac{p}{s}$. Consider the following, slightly modified algorithm:

```
Algorithm GR-1dP
run the greedy algorithm up to the point that the fractional object would be included
let $S$ be the subset of objects computed
$\alpha$ := the profit value of the greedy solution $S$
if $p_{max} > \alpha$ then
    return the set with the object of profit $p_{max}$
else
    return $S$
```

Theorem 13.8 *Algorithm GR-1dP computes a feasible solution to the 1-dimensional packing problem within performance ratio* 2 *and a running time of* $O(n \log n)$*.*

Proof: We have to show that GR-1dP returns a solution with a profit value β such that $\beta \geq \frac{1}{2} \cdot OPT$.

Suppose by way of contradiction that $\beta < \frac{1}{2} \cdot OPT$. It means in particular that $\alpha < \frac{1}{2} \cdot OPT$. Note that the packing which would realize OPT is a feasible solution of the fractional packing problem, which thus has an optimum $\geq OPT$. As the greedy algorithm would fill the bin optimally if at most one fractional object were included, it follows that in the assumed case the greedy algorithm must indeed include a fractional object that contributes a profit $> \frac{1}{2} \cdot OPT$ (the whole solution realizes an optimum $\geq OPT$). In particular the object must have a profit value $> \frac{1}{2} \cdot OPT$. Hence $p_{max} > \frac{1}{2} \cdot OPT > \alpha$. But then, by its program, GR-1dP would have returned the object of profit p_{max} and $\beta = p_{max} > \frac{1}{2} \cdot OPT$, contradiction. ■

The construction that proves Theorem 13.6 uses a mixture of the ideas discussed in this lecture.

References

[1] O.H. Ibarra, C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems, *Journal of the ACM* 22 (1975) 463-468.

[2] E. Lawler. Fast approximation algorithms for knapsack problems, *Math of Operations Research* 4 (1979) 339-356.