3 May 2021

Approximation Algorithms via Dynamic Programming (§11.8)

For the knapsack problem, which is NP-Hard to solve exactly, we will see an algorithm that gets within $1+\varepsilon$ factor of the opt solution in running time $\text{poly}\left(\frac{n}{\varepsilon}\right)$, for any $\varepsilon > 0$.

Recall: In the knapsack problem we are given n items. Item i has value v_i and weight w_i.
Total weight constraint: select any subset of items if their total weight is $\leq W$.
(W is given in the input.)
Objective: find set S maximizing $\sum_{i \in S} v_i$ under the total-weight constraint.

We previously saw:
1. If weights are integers, the problem can be solved in $O(nW)$ time using dyn prog.
2. The problem is NP-Hard.

(Note that giving W as input data uses only $O(\log W)$ bits, so nW is potentially exponential in the input size.)

A different dynamic program that works when values (not weights) are small.

Suppose values are integers, and Q is an upper bound on the value of the best knapsack solution.

DYN PROG FOR KNAPSACK

```
for i = 0, 1, 2, ..., n:
    for v = 0, 1, 2, ..., Q:
        if v = 0:
            M[i,v] = 0          // M[i,v] will store the min
                                   weight of a subset of {1,...,i}
            S[i,v] = ∅             that attains value ≥ v.
        if i = 0 and v > 0:        If no such subset,
            M[i,v] = ∞             M[i,v] = ∞.
        if i > 0:               // S[i,v] = the set that attains
                                      M[i,v].
            v' = max{v - v_i, ∅}   // if item i is chosen, then
                                      remaining elements must
                                      have values summing to ≥ v'.
```

$M[i,v] = \min\{ M[i-1,v],\ M[i-1,v'] + w_i \}$

if $M[i-1,v] \leq M[i-1,v'] + w_i$:

$S[i,v] = S[i-1,v]$

else:

$S[i,v] = S[i-1,v'] \cup \{i\}$

endfor

endfor

// for all i, v, $0 \leq i \leq n$, $0 \leq v \leq Q$, $M[i,v]$ now stores the min weight of a subset of $\{1, \ldots, i\}$ whose combined value is $\geq v$.

Let $v^* = \max\{ v \mid M[n,v] \leq W \}$

Output $S[n,v^*]$.

RUNNING TIME: $O(n \cdot Q)$.

(Recall values are integers, Q = an upper bound on the val. of opt knapsack solution.)

Observation: If values are all multiples of a common divisor, b, and the value of opt solution is $\leq Q \cdot b$, we can solve in $O(n \cdot Q)$.

Reason: Just replace each value v_i with $v_i' = v_i/b$. These are all integers. Now use the alg. above on the rescaled values.

Approximating the opt. solution to the knapsack problem in general:

0. Eliminate all items with $w_i > W$.

1. Choose a parameter, b.
(We will be setting $b = \frac{\varepsilon}{2n} \cdot \max_i \{v_i\}$.
The reason for choosing this b will become apparent at the end.)
When designing alg's like this, leave b as an undefined parameter, perform the analysis, and choose value of b at the end of the analysis.

2. Round up each value v_i to the next larger multiple of b.

$$\tilde{v}_i := \lceil v_i/b \rceil \cdot b$$

3. Choose a number, Q, such that optimum knapsack solution with values $\tilde{v}_i$ is guaranteed to be at most $Q \cdot b$.

Set $Q = \sum_{i=1}^{n} \lceil v_i / b \rceil$.

Because then

$$Q \cdot b = \sum_{i=1}^{n} \left\lceil \frac{v_i}{b} \right\rceil \cdot b = \sum_{i=1}^{n} \tilde{v}_i$$

4. Run the algorithm above on input data $(\tilde{v}_i, w_i)$, W, Q. to solve this instance of the knapsack problem in time $O(n \cdot Q)$.

5. Output that set as our approximately optimal knapsack solution. Call the set $\tilde{S}$.

"Making a small change to the input values, shouldn't change the value of the opt solution very much."

Observation: Let $v(S) = \sum_{i \in S} v_i$

$$\tilde{v}(S) = \sum_{i \in S} \tilde{v}_i .$$

Since $v_i \leq \tilde{v}_i < v_i + b$ for all i,

$$v(S) \leq \tilde{v}(S) < v(S) + nb \quad \text{for all sets } S.$$

Let S^* be the opt solution of the original knapsack problem (with values v_i).
Recall $\tilde{S}$ is optimal for values $\tilde{v}_i$.

Hence

$$v(S^*) \leq \tilde{v}(S^*) \leq \tilde{v}(\tilde{S}) \leq v(\tilde{S}) + nb.$$

Recall

$$b = \frac{\varepsilon}{2n} \max_i \{v_i\}.$$
$$\leq \frac{\varepsilon}{2n} v(S^*)$$

S^* has at least as much value as $\{i\}$ for all i.

$$nb \leq \frac{\varepsilon}{2} v(S^*)$$

So $v(S^*) \leq \tilde{v}(\tilde{S}) \leq \left(1 + \frac{\varepsilon}{2}\right) v(S^*)$.

What about $v(\tilde{S})$?

$$v(\tilde{S}) \geq \tilde{v}(\tilde{S}) - bn$$
$$\geq v(S^*) - bn$$
$$\geq v(S^*) - \frac{\varepsilon}{2} v(S^*)$$

Conclusion. We found a set $\tilde{S}$ s.t.

$$\left(1 - \frac{\varepsilon}{2}\right) v(S^*) \leq v(\tilde{S}) \leq v(S^*).$$

Approximates the opt within better than $1-\varepsilon$ factor.

Running time? $O(n \cdot Q) = O\left(\frac{n^3}{\varepsilon}\right)$.

What is Q?

$$Q = \sum_{i=1}^{n} \left\lceil \frac{v_i}{b} \right\rceil,$$

$$b = \frac{\varepsilon}{2n} \max_i \{v_i\}.$$

$$\frac{1}{b} \leq \frac{2n}{\varepsilon} \cdot \frac{1}{v_i} \quad \forall i$$

$$\frac{v_i}{b} \leq \frac{2n}{\varepsilon} \quad \forall i$$

$$\left\lceil \frac{v_i}{b} \right\rceil \leq \frac{2n}{\varepsilon} + 1.$$

$$Q \leq \frac{2n^2}{\varepsilon} + n \leq O\left(\frac{n^2}{\varepsilon}\right).$$