30 April 2018

Recap: Knapsack items have size s_i, value v_i.
Choose subset to maximize $\sum_{i \in S} v_i$ while $\sum_{i \in S} s_i \le B$.

Dynamic program with running time $O(nV)$ where $V = \sum_{i=1}^{n} v_i$, assuming all v_i are integers.

Approx. algorithm: Keep sizes (s_i) and budget (B) the same.
Modify values to

$$\tilde{v}_i = \lceil \varepsilon \cdot v_i \rceil$$

where $\varepsilon > 0$ is a parameter to be fixed later.
Solve modified problem, output optimal set for $(s_i, \tilde{v}_i)$.

Running time $O(\varepsilon n V + n^2)$.

If ε is very close to zero, the running time will be very good, i.e. $O(n^2)$, but the approximation might be terrible.

Reasoning about the approximation error...

$$\frac{1}{\varepsilon} \tilde{v}_i \ge v_i \ge \frac{1}{\varepsilon}(\tilde{v}_i - 1)$$

If S denotes the output of our algorithm, which maximizes $\sum \tilde{v}_i$, and S^* denotes the optimum knapsack solution, which maximizes $\sum v_i$,

$$\sum_{i \in S} v_i \ge \frac{1}{\varepsilon} \sum_{i \in S} (\tilde{v}_i - 1) \ge \left(\frac{1}{\varepsilon} \sum_{i \in S} \tilde{v}_i \right) - \frac{n}{\varepsilon}$$

Correctness guarantee of dyn prog running on $\tilde{v}_i$ instance.

$$\ge \left(\frac{1}{\varepsilon} \sum_{i \in S^*} \tilde{v}_i \right) - \frac{n}{\varepsilon} \ge \sum_{i \in S^*} v_i - \frac{n}{\varepsilon}.$$

If we want S to be a $(1+\delta)$-approximation to the optimum, we want to choose ε such that

$$(1+\delta) \sum_{i \in S} v_i \ge \sum_{i \in S^*} v_i \qquad \text{(def'n of approximation factor)}$$

According to the inequality above, it suffices to ensure that

$$(1+\delta)\left(\sum_{i \in S^*} v_i - \frac{n}{\varepsilon} \right) \ge \sum_{i \in S^*} v_i$$

$$(1+\delta)\left(\sum_{i\in S^*} v_i - \frac{n}{\varepsilon}\right) \geq \sum_{i\in S^*} v_i$$

Solving for ε...

$$\delta\left(\sum_{i\in S^*} v_i\right) - \frac{(1+\delta)n}{\varepsilon} \geq 0$$

$$\delta\left(\sum_{i\in S} v_i^*\right) \geq \frac{(1+\delta)n}{\varepsilon}$$

$$\varepsilon \geq \frac{(1+\delta)n}{\delta\left(\sum_{i\in S^*} v_i\right)}$$

We could use this ε if we knew how to calculate the RHS before running the dyn prog algorithm.

But the approx works as long as $\varepsilon \geq$ RHS.
We just need an upper bound on RHS, which entails finding a lower bound on the denominator, e.g. the value of any <u>suboptimal</u> knapsack solution.

E.g. V_{max}, the max value of any individual element, is a lower bound on $\sum_{i\in S^*} v_i$. Because the opt knapsack solution has at least as much value as the best singleton set.

So, put $\varepsilon := \frac{(1+\delta)n}{\delta\, V_{max}}$ at the start, then compute $\tilde{v}_i = \lceil \varepsilon \cdot v_i \rceil$ and run the dyn prog to get a knapsack solution.

Approx guarantee : $1+\delta$ factor.
Running time:

$$\boxed{V = \sum_{i=1}^{n} v_i \leq n \cdot v_{max}}$$

$$O(\varepsilon n V + n^2) = O\left(\frac{(1+\delta)n^2}{\delta}\frac{V}{v_{max}} + n^2\right)$$

$$= O\left(\frac{1}{\delta}\cdot n^3\right)$$

Linear Programming: Applies to broader set of problems than dynamic programming, but tends to give worse approximations.

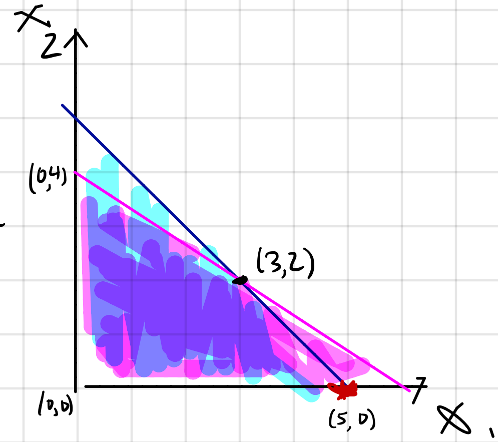
Idea: represent the NP-hard problem as choosing an integer vector to optimize a linear function under some constraints. Instead choose an optimal fractional vector, then round it.

Linear program: choose a vector $\vec{x} \in \mathbb{R}^n$ to maximize a linear function, subject to some linear inequality constraints.

E.g. maximize $3x_1 + 2x_2$

subject to

$$x_1 + x_2 \leq 5$$
$$2x_1 + 3x_2 \leq 12$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$



Maximum of 15 is attained at (5,0).

In high dimensions the feasible region has exponentially many vertices, so brute force search is infeasible.

But there exist poly-time algorithms for linear programming.