1 How Good Can Approximation Ratios Get?

How “hard” are hard problems?

• All approx ratios we have seen so far are at least some constant.
• Wouldn’t it be nice if we could improve the ratio by working harder?
• For some (but not all) NP optimization problems, this is possible.
• Let $Q$ be an NP optimization problem.
• Let $A(x, \epsilon)$ be an algorithm for $Q$ that takes in an instance $x$ of $Q$ and an extra parameter $\epsilon > 0$.
• Let $v^*$ be value of optimal solution to $x$, and let $v_\epsilon$ be value of solution produced by calling $A(x, \epsilon)$.

- **Defn:** $A$ is called a polynomial-time approximation scheme (PTAS) for $Q$ if
  1. $A$ runs in time polynomial in $|x|$.
  2. $\frac{v_\epsilon}{v^*} \leq 1 + \epsilon$ (for minimization), or
  3. $\frac{v^*}{v_\epsilon} \leq 1 + \epsilon$ (for maximization).

• Intuitively, the smaller we make $\epsilon$, the better our approximation.
• No guarantee that $A$’s time doesn’t scale exponentially, or even worse, with decreasing $\epsilon$!!

- **Defn:** $A$ is called a fully polynomial-time approximation scheme (FPTAS) for $Q$ if
  1. $A$ is a PTAS for $Q$.
  2. $A$ runs in time polynomial in both $|x|$ and $\frac{1}{\epsilon}$.
• An FPTAS doesn’t blow up as $\epsilon$ gets smaller.

- **Note:** even an FPTAS’s running time is polynomial in $1/\epsilon$, that is, pseudopolynomial in the size of (i.e. number of digits in) $1/\epsilon$.

- **Claim:** We can’t have a PTAS that runs in time $\log(1/\epsilon)$, i.e. actually polynomial in the size of $1/\epsilon$, unless $P=NP$. 

• **Pf:** Suppose that for NP-optimization problem \( Q \), which is a minimization, we have a PTAS that runs in time polynomial in both the size of its input and \( \log(1/\epsilon) \).

• Let \( v^* \) be the value of an optimal solution to some instance \( x \) of \( Q \).

• Suppose that \( v^* \) is an integer (if it’s rational, multiply all values in the problem by a “big enough” number), and that \( v^* \) is at most \( f(|x|) \), where \( f(n) = O(2^{\text{poly}(n)}) \).

• (If not, it would take more than polynomial time in \( |x| \) just to write down \( v^* \), and so there can be no polynomial-time algorithm for \( Q \) at all).

• Given \( x \), set \( \epsilon = \frac{1}{2f(|x|)} \), and run the PTAS.

• The resulting solution has objective value

\[
v_\epsilon \leq (1 + \epsilon) v^* = \left(1 + \frac{1}{2f(|x|)}\right) v^* \leq \left(1 + \frac{1}{2v^*}\right) v^* = v^* + 1/2.
\]

• Since \( v^* \) and \( v_\epsilon \) are integers, \( v_\epsilon = v^* \).

• Moreover, the FPTAS runs in time polynomial in both \( |x| \) and \( \log(f(|x|)) \), which by assumption is polynomial in \( |x| \).

• Hence, we can find the optimal objective for any instance \( Q \) in polynomial time, and so P=NP. QED

2 Knapsack the Hard Way

Next up, we give an FPTAS for the 0-1 knapsack problem!

• We begin with following pseudopolynomial DP algorithm to solve 0-1 knapsack.

• Let \((S, W)\) be an instance of 0-1 knapsack.

• Item \( i \) in \( S \) has weight \( w_i \) and value \( v_i \).

• **Question:** what is weight of lightest subset of items with total value = \( v \)?

• **First choice:** opt soln either includes last item, or it does not.

• **Inductive Structure:**
  - If last item chosen, we seek lightest subset of items \( 1 \ldots n - 1 \) with total value \( v - v_n \).
  - Else, we seek lightest subset of items \( 1 \ldots n - 1 \) with total value \( v \).

• **Optimal Substructure:** Let \( w' \) be weight of opt soln to subproblem.
– If last item is used, total weight is \(w' + w_n\).
– If last item is not used, total weight is \(w'\).

• Let \(W_{i,v}\) be weight of lightest subset of items \(1\ldots i\) with value \(v\).

• Following recurrence computes \(W_{i,v}\):
  \[
  W_{i,v} = \min (W_{i-1,v}, W_{i-1,v-v_i} + w_i)
  \]

• **Base cases:** \(W_{0,*} = \infty\) except for \(W_{0,0} = 0, W*, 0 = 0\)

• **Subproblem ordering:** by increasing \(i\), and by increasing \(v\) within each \(i\).

So far, so good. But how does this help solve knapsack?

• **Obs 1:** Let \(V = \max_{i=1}^n v_i\); then the maximum possible knapsack value is \(\leq nV\). Hence, we can compute \(W_{i,v}\) for \(0 \leq i \leq n\) and \(0 \leq v \leq nV\) in time \(O(n^2V)\).

• **Obs 2:** For each possible \(v\), looking at \(W_{n,v}\) tells us whether any subset of items \(1\ldots n\) achieves value \(v\) with total weight < \(W\).

• Hence, in additional time \(O(nV)\), we can find a knapsack with total weight \(\leq W\) and maximum value!

• Call this algorithm VBK, for “value-bound knapsack”.

• VBK runs in time \(O(n^2V)\), which is pseudopolynomial in the item values, but not in their weights or the capacity!

• (Note that we can backtrack as usual, in time \(O(n)\), to reconstruct the lightest knapsack for a given \(v\).)

3 And Now, A Sneaky Trick

We will use VBK to build an FPTAS for 0-1 knapsack.

• Algorithm FP-KNAPSACK\((S,W,\epsilon)\) runs as follows on input with \(n\) items.

• Let \(V = \max_{i=1}^n v_i\) as above.

• Set
  \[
  K = \frac{V}{n (1 + \frac{1}{\epsilon})}.
  \]
  (Where does this \(K\) come from? Stay tuned!)

• Form a new knapsack instance \((S',W')\) from \((S,W)\) by replacing value \(v_i\) of every item \(i\) with
  \[
  v'_i = \lfloor v_i/K \rfloor.
  \]

• (Note: as \(\epsilon \to 0\), \(K\) gets very small for any fixed \(n\); if \(K \leq 1\), we might as well use full VBK without any rounding. However, for any fixed \(\epsilon\), you will eventually round given a large enough \(n\).)
• Run VBK on \((S', W)\) and find an optimal item set \(\pi\).
• Return \(\pi\).

Is this algorithm any good?

• Firstly, FP-KNAPSACK returns a feasible knapsack, since modified instance has same item weights and same capacity as original.
• Secondly, observe that largest item value in \((S', W)\) is now only \(V/K\).
• Hence, the running time of VBK, which dominates the time of FP-KNAPSACK, is

\[
O(n^2V/K) = O\left(n^3\left(1 + \frac{1}{\epsilon}\right)\right)
\]

which is polynomial in both \(n\) and \(1/\epsilon\).

So far so good, but what about the approximation ratio?

• **Claim**: FP-KNAPSACK yields a \(1 + \epsilon\) approximation.
• **Pf**: Let \(\pi^*\) be an optimal soln to instance \((S, W)\), and let \(\pi_\epsilon\) be solution found by FP-KNAPSACK with parameter \(\epsilon\).
• Let \(v^*\) and \(v_\epsilon\) be the values of \(\pi^*, \pi_\epsilon\) with respect to instance \((S, W)\).
• Finally, let \(v'\) be value of \(\pi_\epsilon\) w/r to the scaled instance \((S', W)\).
• We first claim that FP-KNAPSACK’s scaled solution does not give an arbitrarily worse result than the optimum.
• In particular, if \(K\) is the scaling factor computed by FP-KNAPSACK, then

\[
v^* - Kv' \leq Kn.
\]
• Suppose not. If items in \(\pi^*\) have values \(\{v_1, \ldots, v_t\}\) in \((S, W)\), then these items in the scaled instance \((S', W)\) have values \(\{[v_1/K], \ldots, [v_t/K]\}\).
• Total value of \(\pi^*\) in \((S', W)\) is therefore

\[
\sum_{i=1}^{t} \frac{v_i}{K} \geq \sum_{i=1}^{t} (\frac{v_i}{K} - 1) = \frac{v^*}{K} - t \geq \frac{v^*}{K} - n.
\]
• But by our assumption, \(v^* - Kv' > Kn\), which implies

\[
\frac{v^*}{K} - n > v',
\]

meaning that the value of \(\pi^*\) in \((S', W)\) is better than the optimum \(\pi_\epsilon\) found by VBK.
• Conclude that indeed, $v^* - Kv' \leq Kn$. (QED for claim)

• Now $v_e \geq Kv'$, since each elt in $\pi_e$ had its value divided by $K$ and then had the floor taken.

• Applying our claim above, we have that

$$v^* - v_e \leq Kn,$$

that is, the algorithm’s solution is at most $Kn$ worse than opt.

• Doing a bit of algebra, we have

$$\frac{v^*}{v_e} \leq \frac{v_e + Kn}{v_e} \leq 1 + \frac{Kn}{v_e} \leq 1 + \frac{Kn}{v^* - Kn}.$$  

• Finally, observe that $v^* \geq V$, since we can always create a knapsack containing the single most valuable item.

• Hence,

$$\frac{v^*}{v_e} \leq 1 + \frac{Kn}{V - Kn} \leq 1 + \epsilon.$$