Chapter 10
Approximation Algorithms III

NEW CS 473: Theory II, Fall 2015
September 24, 2015

10.1 Subset Sum

10.1.0.1 Subset Sum

**Subset Sum**

**Instance:** $X = \{x_1, \ldots, x_n\} - n$ integer positive numbers, $t$ - target number

**Question:** $\exists$ subset of $X$ s.t. sum of its elements is $t$?

Assume $x_1, \ldots, x_n$ are all $\leq n$. Then this problem can be solved in

(A) The problem is still **NP-Hard**, so probably exponential time.

(B) $O(n^3)$.

(C) $2^{O(\log^2 n)}$.

(D) $O(n \log n)$.

(E) None of the above.

10.1.0.2 Subset Sum

**Subset Sum**

**Instance:** $X = \{x_1, \ldots, x_n\} - n$ integer positive numbers, $t$ - target number

**Question:** $\exists$ subset of $X$ s.t. sum of its elements is $t$?

---

$M$: Max value input numbers.

R.T. $O(Mn^2)$.

---

**SolveSubsetSum** ($X$, $t$, $M$)

$b[0 \ldots Mn] \leftarrow$ false

// $b[x]$ is true if $x$ can be

// realized by subset of $X$.

$b[0] \leftarrow$ true.

for $i = 1, \ldots, n$ do

for $j = Mn$ down to $x_i$ do

$b[j] \leftarrow B[j - x_i] \lor B[j]$

return $B[t]$
10.1.1 Subset Sum

10.1.1.1 Efficient algorithm???

(A) Algorithm solving Subset Sum in $O(Mn^2)$.
(B) $M$ might be prohibitly large...
(C) if $M = 2^n \Rightarrow$ algorithm is not polynomial time.
(D) Subset Sum is NPC.
(E) Still want to solve quickly even if $M$ huge.
(F) Optimization version:

**Subset Sum Optimization**

| Instance: $(X, t)$: A set $X$ of $n$ positive integers, and a target number $t$. |
| Question: The largest number $\gamma_{opt}$ one can represent as a subset sum of $X$ which is smaller or equal to $t$. |

10.1.2 Subset Sum

10.1.2.1 2-approximation

**Lemma 10.1.1.** (A) $(X, t)$; Given instance of Subset Sum. $\gamma_{opt} \leq t$: Opt.
(B) $\Rightarrow$ Compute legal subset with sum $\geq \gamma_{opt}/2$.
(C) Running time $O(n \log n)$.

**Proof:**
(A) Sort numbers in $X$ in decreasing order.
(B) Greedily - add numbers from largest to smallest (if possible).
(C) $s$: Generates sum.
(D) $u$: First rejected number. $s'$: sum before rejection.
(E) $s' > u > 0, s' < t$, and $s' + u > t \Rightarrow t < s' + u < s' + s' = 2s' \Rightarrow s' \geq t/2$.

10.1.3 On the complexity of $\varepsilon$-approximation algorithms

10.1.3.1 Polynomial Time Approximation Schemes

**Definition 10.1.2 (PTAS).** PROB: Maximization problem.

$\varepsilon > 0$: approximation parameter.

$\mathcal{A}(I, \varepsilon)$ is a polynomial time approximation scheme (PTAS) for PROB:

(A) $\forall I$: $(1 - \varepsilon)\text{opt}(I) \leq \mathcal{A}(I, \varepsilon) \leq \text{opt}(I)$,
(B) $|\text{opt}(I)|$: opt price,
(C) $|\mathcal{A}(I, \varepsilon)|$: price of solution of $\mathcal{A}$.
(D) $\mathcal{A}$ running time polynomial in $n$ for fixed $\varepsilon$.

For minimization problem: $|\text{opt}(I)| \leq |\mathcal{A}(I, \varepsilon)| \leq (1 + \varepsilon)|\text{opt}(I)|$.

10.1.3.2 Polynomial Time Approximation Schemes

(A) Example: Approximation algorithm with running time $O(n^{1/\varepsilon})$ is a PTAS.

Algorithm with running time $O(1/\varepsilon^n)$ is not.
(B) Fully polynomial...
Definition 10.1.3 (FPTAS). An approximation algorithm is *fully polynomial time approximation scheme* (FPTAS) if it is a PTAS, and its running time is polynomial both in \( n \) and \( 1/\varepsilon \).

(C) Example: PTAS with running time \( O(n^{1/\varepsilon}) \) is not a FPTAS.
(D) Example: PTAS with running time \( O(n^2/\varepsilon^3) \) is a FPTAS.

10.1.3.3 Approximating Subset Sum

**Subset Sum Approx**

**Instance:** \((X, t, \varepsilon)\): A set \( X \) of \( n \) positive integers, a target number \( t \), and parameter \( \varepsilon > 0 \).

**Question:** A number \( z \) that one can represent as a subset sum of \( X \), such that \((1 - \varepsilon)\gamma_{opt} \leq z \leq \gamma_{opt} \leq t\).

10.1.4 Approximating Subset Sum

10.1.4.1 Looking again at the exact algorithm

**ExactSubsetSum**\((S, t)\)

\[
\begin{align*}
n & \leftarrow |S| \\
P_0 & \leftarrow \{0\} \\
& \text{for } i = 1 \ldots n \text{ do} \\
& \quad P_i \leftarrow P_{i-1} \cup (P_{i-1} + x_i) \\
& \quad \text{Remove from } P_i \text{ all elements } > t \\
& \text{return largest element in } P_n
\end{align*}
\]

(A) \( S = \{a_1, \ldots, a_n\} \)
\[
x + S = \{a_1 + x, a_2 + x, \ldots a_n + x\}
\]

(B) Lists might explode in size.

10.1.4.2 Trim the lists...

\( L' \): Inc. sorted list of numbers

**Trim**\((L', \delta)\)

\[
\begin{align*}
L & = \langle y_1 \ldots y_m \rangle \\
curr & \leftarrow y_1 \\
L_{out} & \leftarrow \{y_1\} \\
& \text{for } i = 2 \ldots m \text{ do} \\
& \quad \text{if } y_i > curr \cdot (1 + \delta) \\
& \quad \quad \text{Append } y_i \text{ to } L_{out} \\
& \quad curr \leftarrow y_i \\
& \text{return } L_{out}
\end{align*}
\]

Definition 10.1.4. For two positive real numbers \( z \leq y \), the number \( y \) is a \( \delta \)-approximation to \( z \) if \( \frac{y}{1 + \delta} \leq z \leq y \).

Observation 10.1.5. If \( x \in L' \) then there exists a number \( y \in L_{out} \) such that \( y \leq x \leq y(1 + \delta) \), where \( L_{out} \leftarrow \text{Trim}(L', \delta) \).
10.1.4.3 Trim the lists...

**ApproxSubsetSum(S, t)**

// $S = \{x_1, \ldots, x_n\}$,
// $x_1 \leq x_2 \leq \ldots \leq x_n$

$n \leftarrow |S|$,
$L_0 \leftarrow \{0\},$

$\delta = \varepsilon/2n$

for $i = 1 \ldots n$ do

$E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$

$L_i \leftarrow \text{Trim}(E_i, \delta)$

Remove from $L_i$ elems $> t$.

return largest element in $L_n$

---

$E_i$: Computed by merging two sorted lists in linear time.

10.1.4.4 Understanding trimming

---

**Trim(L', \delta)**

$L = (y_1 \ldots y_m)$

curr $\leftarrow y_1$

$L_{out} \leftarrow \{y_1\}$

for $i = 2 \ldots m$ do

if $y_i > curr \cdot (1 + \delta)$

Append $y_i$ to $L_{out}$

curr $\leftarrow y_i$

return $L_{out}$

---

...
10.1.4.5 Remark

(A) Can assume that trimmed lists $L_i$ are sorted...
(B) Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
(C) So, this is just copy, shift, and merge of two sorted lists.
(D) ... resulting in a sorted list.
(E) takes linear time in size of lists.

10.1.4.6 Analysis

(A) $E_i$ list generated by algorithm in $i$th iteration.
(B) $P_i$: list of numbers (no trimming).

Claim 10.1.6. For any $x \in P_i$ there exists $y \in L_i$ such that $y \leq x \leq (1 + \delta)^i y$.

Proof

(A) If $x \in P_1$ then follows by observation above.
(B) If $x \in P_{i-1} \implies$ (induction) $\exists y' \in L_{i-1}$ s.t. $y' \leq x \leq (1 + \delta)^{i-1} y'$.
(C) By observation $\exists y \in L_i$ s.t. $y \leq y' \leq (1 + \delta)y$, As such,

\[ y \leq y' \leq x \leq (1 + \delta)^{i-1} y' \leq (1 + \delta)^i y. \]

10.1.4.7 Proof continued

Proof continued

(A) If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
(B) By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1 + \delta)^{i-1} \alpha'$.
(C) Thus, $\alpha' + x_i \in E_i$.
(D) $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1 + \delta)x'$.
(E) Thus, $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1 + \delta)^{i-1} \alpha' + x_i \leq (1 + \delta)^{i-1} (\alpha' + x_i) \leq (1 + \delta)^i x'$. ■

10.1.4.8 Running time

10.1.4.9 Running time of ApproxSubsetSum

Lemma 10.1.7. For $x \in [0, 1]$, it holds $\exp(x/2) \leq (1 + x)$.

Lemma 10.1.8. For $0 < \delta < 1$, and $x \geq 1$, we have

\[ \log_{1+\delta} x \leq \frac{2\ln x}{\delta} = O\left(\frac{\ln x}{\delta}\right). \]

See notes for a proof of lemmas.
10.1.4.10 Running time of ApproxSubsetSum

Observation 10.1.9. In a list generated by Trim, for any number \(x\), there are no two numbers in the trimmed list between \(x\) and \((1 + \delta)x\).

Lemma 10.1.10. \(|L_i| = O \left( \frac{n}{\varepsilon} \log n \right)\), for \(i = 1, \ldots, n\).

10.1.4.11 Running time of ApproxSubsetSum

Proof: (A) \(L_{i-1} + x_i \subseteq [x_i, ix_i]\).
(B) Trimming \(L_{i-1} + x_i\) results in list of size

\[
\log_{1+\delta} \frac{ix_i}{x_i} = O \left( \frac{\ln i}{\delta} \right) = O \left( \frac{\ln n}{\delta} \right),
\]

(C) Now, \(\delta = \varepsilon/2n\), and

\[
|L_i| \leq |L_{i-1}| + O \left( \frac{\ln n}{\delta} \right) \leq |L_{i-1}| + O \left( \frac{n \ln n}{\varepsilon} \right)
\]

\[
= O \left( \frac{n^2 \log n}{\varepsilon} \right).
\]

10.1.4.12 Running time of ApproxSubsetSum

Lemma 10.1.11. The running time of ApproxSubsetSum is \(O \left( \frac{n^3 \log n}{\varepsilon} \right)\).

Proof: (A) Running time of ApproxSubsetSum dominated by total length of \(L_1, \ldots, L_n\).
(B) Above lemma implies \(\sum_i |L_i| = O \left( n \times \frac{n^2}{\varepsilon} \log n \right) = O \left( \frac{n^3 \log n}{\varepsilon} \right)\).
(C) Trim runs in time proportional to size of lists.
(D) Overall, R.T. \(O \left( \frac{n^3 \log n}{\varepsilon} \right)\).

10.1.4.13 ApproxSubsetSum

Theorem 10.1.12. ApproxSubsetSum returns \(u \leq t\), s.t. \(\frac{\gamma_{\text{opt}}}{1+\varepsilon} \leq u \leq \gamma_{\text{opt}} \leq t\), \(\gamma_{\text{opt}}: \text{opt solution.}

Running time is \(O((n^3/\varepsilon) \log n)\).

Proof: (A) Running time from above.
(B) \(\gamma_{\text{opt}} \in P_n\): optimal solution.
(C) \(\exists z \in L_n\), such that \(z \leq \text{opt} \leq (1 + \delta)^n z\)
(D) \((1 + \delta)^n = (1 + \varepsilon/2n)^n \leq \exp \left( \frac{\varepsilon}{2} \right) \leq 1 + \varepsilon\), since \(1 + x \leq e^x\) for \(x \geq 0\).
(E) \(\gamma_{\text{opt}}/(1 + \varepsilon) \leq z \leq \text{opt} \leq t\).
10.2 Maximal matching

10.2.0.1 Maximal matching

(A) $G = (V, E)$
(B) Compute maximal matching...
(C) $X \subseteq E$ which is maximal and independent.
(D) Maximal = can not improved by adding an edge.
(E) Maximum = largest possible set among all possible sets.
(F) Computing the maximum is hard then computing maximal solution.
(G) Q: Find maximal matching quickly and of large size...

10.2.0.2 An example of the greedy algorithm...

\begin{figure}
\centering
\includegraphics[width=\textwidth]{example_graph.png}
\caption{Example graph for maximal matching.}
\end{figure}
10.2.0.3 Maximal matching: Algorithm

(A) Algorithm: Repeatedly pick an arbitrary edge and remove it.
(B) $M$: Generated matching. $X$: Maximal matching.
(C) Clearly a maximal matching...
(D) This is a 2-approximation to the maximum matching.
(E) Because...
(F) Every edge in $M$ “kills” two edges of $X$ in the worst case.

10.2.0.4 Maximal matching: Result

Theorem 10.2.1. Given a graph $G$ one can compute in $O(n + m)$ time, a maximal matching with at least $|X|/2$ edges, where $X$ is the size of the maximum (optimal) matching.

10.2.1 Bin packing

10.2.2 Bin packing

10.2.2.1 Problem definition

**Bin Packing**

**Instance:** $v$: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: $n$ items
\(\alpha_i\): size of $i$th item.

**Target:** Find min $\# B$, and a decomposition $S_1, \ldots, S_B$ of $S$, such that $\forall j \sum_{x \in S_j} \leq v$.

(A) $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$. 

17
10.2.3 Bin packing: First fit

10.2.3.1 Analysis

**Lemma 10.2.2.** First fit is a 2-approximation.

*Proof:* Observe that only one bin can have less than $v/2$ content in it...

10.3 Independent set of axis-parallel rectangles

10.3.0.1 An example

10.3.1 Independent set of rectangles

10.3.1.1 Algorithm: Divide & Conquer
10.3.2 Independent set of rectangles

10.3.2.1 Algorithm: Divide & Conquer

\( \mathcal{R} \): A set of axis parallel rectangles.

\[ \text{RectIndep}(\mathcal{R}) : \]

\[ \text{if } |\mathcal{R}| \leq 10 \text{ then} \]

\[ \text{Solve by brute force} \]

\[ \text{return size of solution} \]

\[ x_M : \text{Median of right } x\text{-coordinate of rects in } \mathcal{R} \]

\[ \ell : \text{Vertical line through } x_M. \]

\[ \mathcal{R}_M : \text{Rects of } \mathcal{R} \text{ intersecting } \ell \]

\[ \mathcal{R}_L, \mathcal{R}_R : \text{Rectangles in } \mathcal{R} \text{ left/ right of } \ell. \]

\[ S_L \leftarrow \text{RectIndep}(\mathcal{R}_L) \]

\[ S_R \leftarrow \text{RectIndep}(\mathcal{R}_R) \]

\[ S_M \leftarrow \text{compute opt solution for } \mathcal{R}_M \text{ (intervals!)} \]

\[ \text{return } \max(S_M, S_L + S_R) \]

10.3.2.2 Analysis

(A) If \( S_M \geq \text{Opt}/(2\lg n) \)... done.

(B) \( \text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2\lg n))\text{Opt}. \)

(C) By induction: \( S_L \geq \text{Opt}_L/(2\lg (n/2)) \) and \( S_R \geq \text{Opt}_R/(2\lg (n/2)). \)

(D) \( S_L + S_R \geq (1 - 1/(2\lg (n/2)))\text{Opt} \)

(E) \[ \frac{1}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)} \]
\[ \frac{2 \lg n - 1}{(2 \lg n)(2 \lg n - 2)} \geq \frac{2 \lg n - 2}{(2 \lg n)(2 \lg n - 2)} \geq \frac{1}{2 \lg n}. \]

(F) Conclude: If \( S_M \leq \text{Opt}/(2 \lg n) \), then \( S_L + S_R \geq \text{Opt}/(2 \lg n) \).

(G) Algorithm is \( 2 \lg n \) approximation.