Fully Polynomial-Time Approximation Scheme for Subset Sum
Lecture notes by Michael Goldwasser

Input: Set $S$ of positive integers $x_1, x_2, \ldots, x_n$ and a target goal $t$.
Decision problem: Is it possible to find a subset of $S$ that sums precisely to $t$?
Optimization problem: What is largest possible subset sum that is at most $t$?

We saw an $O(n \cdot t)$ algorithm that solves the decision problem, and which can easily be adapted to solve the optimization problem (we’ll review below). But this is not formally a polynomial algorithm because the problem input size is $O(n \cdot \log t)$.

Unless P = NP, there cannot be a polynomial-time solution. But we will see that we can get a polynomial time algorithm that guarantees to find a solution that is at most a factor of $(1 + \epsilon)$ away from the optimal solution for any constant $\epsilon > 0$. The catch is that the runtime depends on epsilon, specifically, $O(\frac{1}{\epsilon} \cdot n^2 \cdot \log t)$. So the closer you want to guarantee you are to the optimal solution, the more expensive the algorithm becomes, and you would need to get to $\epsilon < \frac{1}{t}$ to be sure that error is strictly less than 1, yet then runtime is back to dependence on $t$.

Since Chapter 35.5 of CLRS provides formal writeup, we get to instead frame the big picture in our presentation.

Exact Algorithm

We wish to build a list $P$ of all sums that can be formed from subsets of $S$. We can build this iteratively by computing $P_i$ which is such a list using only subsets of $\{x_1, x_2, \ldots, x_i\}$, and thus $P = P_n$. Given $P_{i-1}$ we can form $P_i$ which consists of everything from $P_{i-1}$ (since we can choose not to use $x_i$ or any sum we can get by adding $x_i$ to any of the totals found in $P_{i-1}$ (we denote this as “$P_i - P_{i-1} + x_i$”).

If we initialize $P_0 = <0>$ and we maintain each in sorted order, we can easily merge sequences $P_{i-1}$ and $P_{i-1} + x_i$ in time linear in the length of the sequence (and we could also throw away any values greater than $t$ as we go). But the problem is that in general it may be that $|P_i| = 2^i$, and so runtime could be $\Theta(2^n)$, or if throwing away large elements, $\Theta(n \cdot t)$, which is not polynomial in the input size.

Example: $S = \{1, 4, 5\}$.

$L_0 = <0>$
$L_1 = <0, 1>$
$L_2 = <0, 1, 4, 5>$
$L_3 = <0, 1, 4, 5, 6, 9, 10>$

Approximation Algorithm

The key insight will be a subroutine to “trim” our list of values at each stage based on a trimming parameter $\delta$ with $0 < \delta < 1$. Subroutine $\text{Trim}(L, \delta)$ will reduce list $L$ of integers to a subsequence $L'$ while guaranteeing that for any $y \in L$ there remains some $z \in L'$ such that $\frac{y}{1+\delta} \leq z \leq y$. In effect $z$ becomes a nearby substitute for removed $y$. 

1
As an example, with $\delta = 0.1$ and $L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$ we might trim to $L' = \langle 10, 12, 15, 20, 23, 29 \rangle$. Notice that removed 11 has a nearby substitute in 10 as $11 \leq 10 \leq 11$. Similarly, elements 21 and 22 are sufficiently approximated by 20, and removed 24 is approximated by remaining 23.

Assume we maintain $L = \langle y_1, y_2, \ldots, y_m \rangle$ in sorted order such that $y_1 < y_2 < \cdots < y_m$. We can implement the following strategy for trimming in $O(m)$ time.

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

$L' = \langle y_1 \rangle$

last = $y_1$

for $j = 2$ to $m$

if $y_j >$ last $\cdot (1 + \delta)$

append $y_j$ to $L'$

last = $y_j$

return $L'$

Note as well that if we only keep values $t$ or less in result $L'$, then $|L'| \leq 2 + \log(1+\delta) t$, because each pair of remaining elements $z < z'$ we have separation such that $z' > (1 + \delta) z$.

Our overall approximation algorithm is as follows for some $0 < \epsilon < 1$:

**Approx-Subset-Sum($S, t, \epsilon$)**

$n = |S|$

$L_0 = \langle 0 \rangle$

for $i = 1$ to $n$

$L_i = \text{Merge}(L_{i-1}, L_{i-1} + x_i)$

$L_i = \text{Trim}(L_i, \frac{\epsilon}{2n})$

remove form $L_i$ any values that are strictly greater than $t$

return largest value in $L_n$

Before proving that this algorithm provides a polynomial-time approximation scheme, we cite some underlying mathematical facts about logs and exponents for $x > 0$.

Fact 1: $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$

Fact 2: $\lim_{n \to \infty} (1 + \frac{x}{n})^n = e^x$

Fact 3: $\ln(1 + x)$ satisfies $\frac{x}{1+x} \leq \ln(1 + x) \leq x$

Let’s first argue that the running time of the proposed algorithm is $O(\frac{1}{\epsilon} \cdot n^2 \cdot \ln t)$. By earlier argument, and given choice of $\delta = \frac{\epsilon}{2n}$, we have that the size of any $L_i$ is

$$|L_i| \leq 2 + \log(1+\frac{\epsilon}{2n}) \cdot t = 2 + \frac{\ln t}{\ln (1 + \frac{\epsilon}{2n})}$$

$$\leq 2 + \frac{2n}{\epsilon} \cdot \left(1 + \frac{\epsilon}{2n} \right) \cdot \ln t = 2 + \frac{2n + \epsilon}{\epsilon} \cdot \ln t$$

$$\leq 2 + \frac{3n}{\epsilon} \cdot \ln t = O(\frac{1}{\epsilon} \cdot n \cdot \ln t)$$

And thus are overall algorithm does $n$ passes each of which is linear in the list length.
**Lemma.** For any \( y \in P_i \), there exists \( z \in L_i \) such that

\[
\frac{y}{(1 + \frac{\epsilon}{2n})} \leq z \leq y
\]

**Proof.** Induction on \( i \) \hfill \Box

**Theorem.** If \( y^* \) is true optimal sum and \( z^* \) is value returned by the algorithm, then \( \frac{y^*}{z^*} \leq 1 + \epsilon \).

**Proof.** By Lemma,

\[
\frac{y^*}{z^*} \leq \left(1 + \frac{\epsilon}{2n}\right)^n.
\]

By Fact 2, we get that

\[
\lim_{n \to \infty} \left(1 + \frac{\epsilon}{2n}\right)^n = e^{\epsilon/2}.
\]

Furthermore, by derivative we find that expression \((1 + \frac{\epsilon}{2n})^n\) is strictly increasing and thus we approach the limit from below and for fixed \( n \) we have

\[
\left(1 + \frac{\epsilon}{2n}\right)^n \leq e^{\epsilon/2}
\]

\[
\leq 1 + \frac{\epsilon}{2} + \left(\frac{\epsilon}{2}\right)^2 \quad \text{due to Fact 1}
\]

\[
\leq 1 + \epsilon \quad \text{due to } 0 < \epsilon < 1
\]

\hfill \Box