COMP 355
Advanced Algorithms

Subset Sum Approximation
Sections 6.4 & 8.8(KT)
Section 35.5(CLRS)

Rhodes College

1

Polynomial Approximation Schemes

To the control the precision of the approximation.

1. Specify a parameter $\epsilon > 0$ as part of the input to the approximation algorithm
2. Require that the algorithm produce an answer that is within a relative error of ϵ of the optimal solution.

Note: It is understood that as ϵ tends to 0, the running time of the algorithm will increase.

Such an algorithm is called a polynomial approximation scheme.

Ex: Running time = $O(2^{(1/\epsilon)}n^2)$

A fully polynomial approximation scheme is one in which the running time is polynomial in both n and $1/\epsilon$.

Ex. Running time = $O((n/\epsilon)^2)$

2

Subset Sum

Given a set S of positive integers $\{x_1, x_2, \ldots, x_n\}$ and a target value t, and we are asked whether there exists a subset $S' \subseteq S$ that sums exactly to t.

The optimization problem is to determine the subset whose sum is as large as possible but not larger than t.

Suppose we are given $0 < \epsilon < 1$. Let $z^* \leq t$ denote the optimum sum.

Approximation Problem: Return a value $z \leq t$ such that $z \geq z^*(1 - \epsilon)$.

If we think of this as a knapsack problem, we want our knapsack to be within a factor of $(1 - \epsilon)$ of being as full as possible. So, if $\epsilon = 0.1$, then the knapsack should be at least 90% as full as the best possible.

3

Exponential Time Algorithm

Exact Subset Sum

```
Exact_SS(x[1..n], t) {
    L = <0>;
    for i = 1 to n do {
        L = MergeLists(L, L+x[i]);
        remove for L all elements greater than t;
    }
    return largest element in L;
}
```

For example, if S = {1, 4, 6} and t = 8 then the successive lists would be

$L_0 = \langle 0 \rangle$

$L_1 = \langle 0 \rangle \cup \langle 0 + 1 \rangle = \langle 0, 1 \rangle$

$L_2 = \langle 0, 1 \rangle \cup \langle 0 + 4, 1 + 4 \rangle = \langle 0, 1, 4, 5 \rangle$

$L_3 = \langle 0, 1, 4, 5 \rangle \cup \langle 0 + 6, 1 + 6, 4 + 6, 5 + 6 \rangle = \langle 0, 1, 4, 5, 6, 7, 10, 11 \rangle$

The last list would have the elements 10 and 11 removed, and the final answer would be 7.

The algorithm runs in $\Omega(2^n)$ time in the worst case, because this is the number of sums that are generated if there are no duplicates, and no items are removed.

4

Approximation Algorithm

To convert this into an approximation algorithm, we will:

- Introduce a way to “trim” the lists to decrease their sizes.
  - The idea is that if the list L contains two numbers that are very close to one another, e.g. 91,048 and 91,050, then we should not need to keep both of these numbers in the list. One of them is good enough for future approximations.

How much trimming can we allow and still keep our approximation bound?

Furthermore, will we be able to reduce the list sizes from exponential to polynomial?

5

Approximation Algorithm

- The trimming must also depend on ϵ. We select $\delta = \epsilon/n$.
- Note that $0 < \delta < 1$. Assume that the elements of L are sorted. We walk through the list. Let z denote the last untrimmed element in L, and let $y \geq z$ be the next element to be considered.

  If $\frac{(y-z)}{y} \leq \delta$, then we trim y from the list.
- Equivalently, this means that the final trimmed list cannot contain two values y and z such that $(1 - \delta)y \leq z \leq y$.
- We can think of z as representing y in the list.

Example:

Given $\delta = 0.1$ and L = <10, 11, 12, 15, 20, 21, 22, 23, 24, 29>

trimmed list L’ = <10, 12, 15, 20, 23, 29>

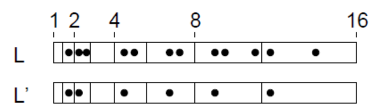
6

Another way to visualize trimming

Another way to visualize trimming is to break the interval from $[1, t]$ into a set of *buckets* of exponentially increasing size. Let $d = 1/(1-\delta)$. Note that $d > 1$. Consider the intervals $[1, d]$, $[d, d^2]$, $[d^2, d^3]$, $\ldots$, $[d^{k-1}, d^k]$ where $d^k \geq t$. If $z \leq y$ are in the same interval $[d^{i-1}, d^i]$ then

$$\frac{y-z}{y} \leq \frac{d^i - d^{i-1}}{d^i} = 1 - \frac{1}{d} = \delta.$$

Thus, we cannot have more than one item within each bucket. We can think of trimming as a way of enforcing the condition that items in our lists are not relatively too close to one another, by enforcing the condition that no bucket has more than one item.



Trimming Lists for Approximate Subset Sum.

Proof

Claim: The number of distinct items in a trimmed list is $O((n \log t)/\epsilon)$, which is polynomial in input size and $1/\epsilon$.

Proof: We know that each pair of consecutive elements in a trimmed list differ by a ratio of at least $d = 1/(1-\delta) > 1$. Let k denote the number of elements in the trimmed list, ignoring the element of value 0. Thus, the smallest nonzero value and maximum value in the trimmed list differ by a ratio of at least d^{k-1}. Since the smallest (nonzero) element is at least as large as 1, and the largest is no larger than t, then it follows that $d^{k-1} \leq t/1 = t$. Taking the natural log of both sides we have $(k-1)\ln d \leq \ln t$. Using the facts that $\delta = \epsilon/n$ and the log identity that $\ln(1+x) \leq x$, we have

$$\begin{aligned} k - 1 &\leq \frac{\ln t}{\ln d} = \frac{\ln t}{-\ln(1-\delta)} \\ &\leq \frac{\ln t}{\delta} = \frac{n \ln t}{\epsilon} \\ k &= O\left(\frac{n \log t}{\epsilon}\right). \end{aligned}$$

Observe that the input size is at least as large as n (since there are n numbers) and at least as large as $\log t$ (since it takes $\log t$ digits to write down t on the input). Thus, this function is polynomial in the input size and $1/\epsilon$.

Approximate Subset Sum

Approximate Subset Sum

```
Trim(L, delta) {
    let the elements of L be denoted y[1..m];
    L' = <y[1]>;                        // start with first item
    last = y[1];                        // last item to be added
    for i = 2 to m do {
        if (last < (1-delta) y[i]) {    // different enough?
            append y[i] to end of L';
            last = y[i];
        }
    }
}

Approx_SS(x[1..n], t, eps) {
    delta = eps/n;                      // approx factor
    L = <0>;                            // empty sum = 0
    for i = 1 to n do {
        L = MergeLists(L, L+x[i]);      // add in next item
        L = Trim(L, delta);             // trim away "near" duplicates
        remove for L all elements greater than t;
    }
    return largest element in L;
}
```

The running time of the procedure is O(n|L|) which is $O(n^2 \ln t/\epsilon)$ by the earlier claim.

9

Approximate Subset Sum Example

For example, consider the set S = {104, 102, 201, 101} and t = 308 and ϵ = 0.20. We have δ = ϵ/4 = 0.05.

$$
\begin{aligned}
\text{init:}\quad & L_0 = \langle 0\rangle \\[1em]
\text{merge:}\quad & L_1 = \langle 0, 104\rangle \\
\text{trim:}\quad & L_1 = \langle 0, 104\rangle \\
\text{remove:}\quad & L_1 = \langle 0, 104\rangle \\[1em]
\text{merge:}\quad & L_2 = \langle 0, 102, 104, 206\rangle \\
\text{trim:}\quad & L_2 = \langle 0, 102, 206\rangle \\
\text{remove:}\quad & L_2 = \langle 0, 102, 206\rangle \\[1em]
\text{merge:}\quad & L_3 = \langle 0, 102, 201, 206, 303, 407\rangle \\
\text{trim:}\quad & L_3 = \langle 0, 102, 201, 303, 407\rangle \\
\text{remove:}\quad & L_3 = \langle 0, 102, 201, 303\rangle \\[1em]
\text{merge:}\quad & L_4 = \langle 0, 101, 102, 201, 203, 302, 303, 404\rangle \\
\text{trim:}\quad & L_4 = \langle 0, 101, 201, 302, 404\rangle \\
\text{remove:}\quad & L_4 = \langle 0, 101, 201, 302\rangle
\end{aligned}
$$

The final output is 302.

The optimum is 307 = 104 + 102 + 101.

Actual relative error in this case is within 2%.

10