Polynomial-time Approximation Schemes

Let Π be a problem in NPO that is NP-hard

- **Suppose, for any $\varepsilon > 0$, we can find a $(1 + \varepsilon)$-approximation algorithm for Π**
  - **really a whole family of algorithms**
  - **one member of the family A_ε for each ε**
  - **complexity must be a polynomial function of the size of the instance (but not necessarily of $1/\varepsilon$)**
- **This is called a *polynomial-time approximation scheme* (*ptas*)**
- **Typically the time complexity for a given ε might be something like $O(n^{1/\varepsilon})$ or $O(2^{1/\varepsilon} n^c)$**
  - **the better the approximation the slower the algorithm**

Subset Sum Problem (SS)

- **Instance:** a sequence $x_1, \ldots, x_n$ of positive integers, and a positive integer k (the target sum)
- **Question:** does there exist a subset S of $\{1, \ldots, n\}$ such that $\sum_{r \in S} x_r = k$?
- **Example instance:** $x_1=7$, $x_2=4$, $x_3=11$, $x_4=4$, $x_5=9$, $x_6=8$; $t = 28$
  - Answer is yes (e.g., $S=\{2,3,4,5\}$) but no if $t=29$

Subset Sum Optimisation Problem (SSO)

- **Instance:** a sequence $x_1, \ldots, x_n$ of positive integers, and a positive integer t (the threshold)
- **Feasible solutions:** subsets S of $\{1, \ldots, n\}$ such that $\sum_{i \in S} x_i \le t$
- **Measure:** $\sum_{i \in S} x_i$
- **GOAL:** max

special case of KP
where weights = profits

- Objective of SSO is to find a subcollection of $x_1, x_2, . . . , x_n$ whose sum is as large as possible, but no greater than t
- Example instance: $x_1=7$, $x_2=4$, $x_3=11$, $x_4=4$, $x_5=9$, $x_6=8$; $t = 29$
- An optimal solution is $S=\{2, 3, 4, 5\}$ of measure 28
  - $x_2 + x_3 + x_4 + x_5 = 4 + 11 + 4 + 9 = 28$
- Exercise: prove that SSO is NP-hard
- Our ptas for SSO starts with an (exponential-time) optimising algorithm, and uses a method called *trimming*
- Given a set S of positive integers, and a positive integer y, define $S + y = \{x + y : x \in S\}$
- E.g. $S = \{1, 2, 3, 5, 9\}$, $S + 2 = \{3, 4, 5, 7, 11\}$

An optimising algorithm for SSO

```
s = {0};
for (int i=1;i<=n;i++)
{   s = s ∪ (s + x_i);
    remove from s any x > t;
}
return max s;    // largest element of s
```

- Invariant: after the i^{th} iteration, s consists of all integers $\leq t$ realisable as the sum of zero or more of $x_1, x_2, \ldots, x_i$
- Represent s as an ordered list
  - so union becomes a 'merge', which can be achieved in time proportional to the length of the list
- The size of s after the i^{th} iteration can be as large as 2^i – hence worst-case complexity is exponential-time

Trimming

```
public List<int> trim (float δ, List<int> s)
// assume that s is an ordered list
{   List<int> z = new List<int>();
    int p = s.get(0);
    z.add(p);
    for (int q : s.sublist(1,s.size())
        if ((1-δ)*q > p) // is q sufficiently larger than p?
        {   z.add(q);       // append q to z
            p = q;
        }
    return z;
}
```

Example: Let $\delta = 0.1$ and

S = {10, *11*, 12, 15, 20, *21*, *22*, 23, *24*, 29}

trim(δ, S) gives {10, 12, 15, 20, 23, 29}

A $(1 + \varepsilon)$-approximation algorithm for SSO

Assume some $\varepsilon > 0$ has been given

```
s = {0};
for (int i=1;i<=n;i++)
{   s = s ∪ (s + x_i);
    s = trim(ε/2n, s);
    remove from s any x > t;
}
return max s;
```

Example: $x_1=7$, $x_2=4$, $x_3=11$, $x_4=4$, $x_5=9$, $x_6=8$; $t=29$

Suppose $\varepsilon=3$, so $\varepsilon/2n = 0.25$ and $1-\delta = 0.75$

i=1: S={ 0, 7 }
i=2: S={ 0, 4, 7, 11 }
i=3: S={ 0, 4, 7, 11, 15, *18*, 22 }
i=4: S={ 0, 4, 7, *8*, 11, 15, *19*, 22, *26* }
i=5: S={ 0, 4, 7, *9*, 11, *13*, 15, *16*, *20*, 22, *24*, *31* }
i=6: S={ 0, 4, 7, *8*, 11, *12*, 15, *19*, 22, *23*, *30* }

Trimmed values in italics

Value returned is 22
(Optimal measure is 28)

Facts about the trimming algorithm for SSO

- *Fact 1*: the value returned by the algorithm is certainly attainable as a sum of elements x_i
  – every element of S is so attainable

- *Fact 2*: the value z returned for a given instance y of SSO satisfies $z \geq m^*(y) / (1 + \varepsilon)$

- *Fact 3*: the algorithm is of complexity $O(n^2 \log t / \varepsilon)$ – for a fixed ε this is polynomial in the input size

- We will not prove Facts 2 & 3

- Since the above algorithm is defined for any $\varepsilon > 0$, we have a ptas for SSO

The Class PTAS

- **NPO** problems $\prod$ that admit a *polynomial-time approximation scheme) (ptas)*
  - Example: SSO
- We have **PO $\subseteq$ PTAS $\subseteq$ APX $\subseteq$ NPO**
- Let $\prod$ be a problem in **PTAS**, so that, for each $\varepsilon > 0$, there is a $(1+\varepsilon)$–approximation algorithm A_ε for $\prod$
- Typically the time complexity of A_ε for a given ε might be of the form, say, $O(n^{1/\varepsilon})$ or $O(n^2/\varepsilon)$
- In both cases, improved approximation means slower algorithm
  - in the first case, A_ε is exponential in $1/\varepsilon$
  - in the second case, A_ε is polynomial in $1/\varepsilon$
- The second case gives a better ptas than the first case

The Class FPTAS

- **NPO problems $\prod$ that admit a *fully polynomial-time approximation scheme (fptas)***
  - **i.e. a ptas such that the time complexity of each approximation algorithm A_ε is polynomial not only in the input size of $\prod$ *but also in* $1/\varepsilon$**
- **The subclass consisting of those problems in NPO that can be approximated most effectively**
  - **Example: SSO**
- **Running time of "trimming" ptas is $O(n^2 \log t / \varepsilon)$ for each ε, where n = number of objects and t = threshold**
- **We have PO $\subseteq$ FPTAS $\subseteq$ PTAS $\subseteq$ APX $\subseteq$ NPO**
- **Each of these inclusions is strict if and only if P$\neq$NP**