Chapter 8

Approximation Algorithms II

CS 573: Algorithms, Fall 2014
September 19, 2014

8.1 Max Exact 3SAT

8.1.0.1 3SAT revisited

(A) Instance of **3SAT** is a boolean formula.
(B) Example: $F = (x_1 + x_2 + x_3)(x_4 + \overline{x_1} + x_2)$.
(C) Decision problem = is the formula has a satisfiable assignment.
(D) Optimization version:

Max 3SAT

Instance: A collection of clauses: $C_1, \ldots, C_m$.
Question: Find the assignment to $x_1, ..., x_n$ that satisfies the maximum number of clauses.

(E) **Max 3SAT** is **NP-Hard**
(F) **Max 3SAT** is a ***maximization problem***.

8.1.0.2 Some definitions

Definition 8.1.1. Algorithm Alg for a maximization problem achieves an approximation factor $\alpha \leq 1$ if for all inputs, we have:

$$\frac{\text{Alg}(G)}{\text{Opt}(G)} \geq \alpha.$$

randomized algorithm: it is allowed to consult with a source of random numbers in making decisions.

Definition 8.1.2 (Linearity of expectations.). Given two random variables X, Y (not necessarily independent, we have that $\mathbf{E}\Big[X + Y\Big] = \mathbf{E}\Big[X\Big] + \mathbf{E}\Big[Y\Big]$.

8.1.0.3 Approximating Max3SAT

Theorem 8.1.3. *Expected* $(7/8)$*-approximation to* ***Max 3SAT*** *in polynomial time.*

F *has* m *clauses* $\implies$ *generated assignment satisfies* $(7/8)m$ *clauses in* expectation.

Proof

(A) $x_1, \ldots, x_n$: n variables used.
(B) Randomly and independently assign 0/1 values to $x_1, \ldots, x_n$.
(C) Y_i: indicator variable is 1 $\iff$ ith clause in instance is satisfied.
(D) $Y = \sum_{i=1}^{m} Y_i$: # clauses satisfied.

8.1.0.4 Approximating Max3SAT - proof continued

Proof continued:

(A) Claim: $\mathbf{E}[Y] = (7/8)m$, $m =$ number of clauses.

$$\mathbf{E}\Big[Y\Big] = \mathbf{E}\Bigg[\sum_{i=1}^{m} Y_i\Bigg] = \sum_{i=1}^{m} \mathbf{E}\Big[Y_i\Big]$$

by linearity of expectation.

(B) $\mathbf{Pr}\Big[Y_i = 0\Big] = \dfrac{1}{2} * \dfrac{1}{2} * \dfrac{1}{2} = \dfrac{1}{8}$. $\implies$ $\mathbf{Pr}[Y_i = 1] = \frac{7}{8}$,

$$\mathbf{E}\Big[Y_i\Big] = \mathbf{Pr}\Big[Y_i = 0\Big] * 0 + \mathbf{Pr}\Big[Y_i = 1\Big] * 1 = \frac{7}{8}.$$

$\mathbf{E}[\text{\# of clauses sat}] = \mathbf{E}[Y] = \sum_{i=1}^{m} \mathbf{E}[Y_i] = (7/8)m$. ∎

8.1.1 Approximating Max3SAT

8.1.1.1 Concluding remarks

(A) Algorithm quality independent of opt...
(B) Algorithm is oblivious.
(C) **?** proved that one can do no better; that is, for any constant $\varepsilon > 0$, one can not approximate **3SAT** in polynomial time (unless **P** = **NP**) to within a factor of $7/8 + \varepsilon$.
(D) Amazing that a trivial algorithm like the above is essentially optimal!

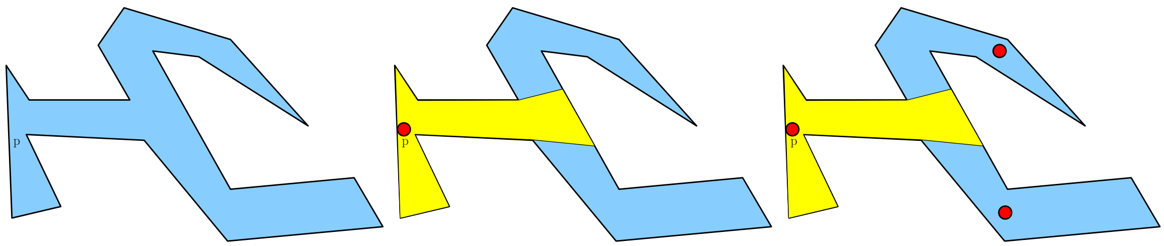
8.1.1.2 Biographical Notes

The Max 3SAT remains hard in the "easier" variant of MAX 2SAT version, where every clause has 2 variables. It is known to be **NP-Hard** and approximable within 1.0741 **?**, and is not approximable within 1.0476 **?**. Notice, that the fact that MAX 2SAT is hard to approximate is surprising as **2SAT** can be solved in polynomial time (!).

8.2 Approximation Algorithms for Set Cover

8.2.1 Guarding an Art Gallery

8.2.1.1 Set cover in the real world



(A) Given: floor plan of an art gallery.
(B) **Target:** Place min # guards that see the whole polygon.
(C) ***Visibility polygon*** at p: region inside polygon that p can see.
(D) Example of **Set Cover**.
(E) **NP-Hard**, no approximation currently known.

8.2.2 Set Cover

8.2.2.1 Set cover

Set Cover

Instance: $(\mathsf{S}, \mathcal{F})$:
S - a set of n elements
$\mathcal{F}$ - a family of subsets of S, s.t. $\bigcup_{X \in \mathcal{F}} X = \mathsf{S}$.
Question: The set $\mathcal{X} \subseteq \mathcal{F}$ such that $\mathcal{X}$ contains as few sets as possible, and $\mathcal{X}$ covers S. Formally, $\bigcup_{X \in \mathcal{X}} X = \mathsf{S}$.

S: ***ground set***
$(\mathsf{S}, \mathcal{F})$: ***set system*** or a ***hypergraph***.
Set Cover is a minimization problem. **NP-Hard**.

8.2.3 Set cover

8.2.3.1 Example

Example 8.2.1. Consider set $\mathsf{S} = \{1, 2, 3, 4, 5\}$ and the family of subsets

$$\mathcal{F} = \Big\{ \{1, 2, 3\},\ \{2, 5\},\ \{1, 4\},\ \{4, 5\} \Big\}.$$

Smallest cover of S is $\mathcal{X}_{opt} = \Big\{ \{1, 2, 3\},\ \{4, 5\} \Big\}$.

8.2.4 Set cover

8.2.4.1 Greedy algorithm

```
GreedySetCover(S, F)
        X_0 ← ∅,    U_0 ← S,    i ← 0
        while U_i is not empty do
            Y_i ← set in F covering largest
                # of elements in U_i
            X_{i+1} ← X_i ∪ {Y_i}
            U_{i+1} ← U_i \ Y_i
            i ← i + 1

        return X_i.
```

(A) S: set of n elements.
(B) $\mathcal{F}$: m sets.
(C) Size of input $\Omega(m+n)$ (and $O(mn)$).

8.2.5 Set cover – Greedy algorithm

8.2.5.1 Analysis

(A) $\mathcal{X}_{opt} = \{V_1, \ldots, V_k\} \subseteq \mathcal{F}$: optimal solution.
(B) U_i: elements not covered in beginning of ith iteration.
(C) $U_1 = \mathsf{S}$.
(D) Y_i: set added to the cover in ith iteration.
(E) $\alpha_i = |Y_i \cap U_i|$: # of new elements being covered.

Claim 8.2.2. *We have* $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_k \geq \ldots \geq \alpha_m$.

Proof: If $\alpha_i < \alpha_{i+1}$ then Y_{i+1} covers more elements than Y_i and we can exchange between them, and get a better set. A contradiction. ■

8.2.6 Set cover – Greedy algorithm

8.2.6.1 Analysis continued

Claim 8.2.3. $\alpha_i \geq |U_i|/k$. *Equivalently:* $|U_{i+1}| \leq (1 - 1/k)\,|U_i|$.

Proof: (A) k: Size of optimal solution.
(B) Opt solution: $\mathcal{O} = \{O_1, \ldots, O_k\}$ covers ground set S.
(C) $\implies \forall i \qquad U_i \subseteq \mathsf{S} \subseteq \bigcup_{i=1}^{k} O_i$ elements of U_i.
(D) $\implies$ one set of opt covers $\geq |U_i|/k$ of U_i.
(E) greedy algorithm picks set Y_i with max cover.
(F) $\implies Y_i$ covers $\alpha_i \geq |U_i|/k$ (prev. not covered) elements.
(G) $|U_{i+1}| = |U_i| - \alpha_i \leq (1 - 1/k)\,|U_i|$.

8.2.7 Set cover – Greedy algorithm

8.2.7.1 Analysis continued

Using the claim $|U_i| \le (1 - 1/k)\,|U_{i-1}| \le (1 - 1/k)^i\,|U_0| = (1 - 1/k)^i n$.

Useful Fact $1 - x \le e^{-x}$.

8.2.8 Set cover – Greedy algorithm

8.2.8.1 Analysis continued

Theorem 8.2.4. **GreedySetCover$(\mathsf{S}, \mathcal{F})$** *generates a cover of* S *using at most* $O(k \log n)$ *sets of* $\mathcal{F}$*, k: size of the cover in opt solution.* $n = |\mathsf{S}|$

Proof: In what round M is U_M empty?

For $M = \lceil 2k \ln n \rceil$: $|U_M| \le \left(1 - \frac{1}{k}\right)^M n \le \exp\!\left(-\frac{1}{k} M\right) n$

$= \exp\!\left(-\frac{\lceil 2k \ln n \rceil}{k}\right) n \le \exp(-2 \ln n)\, n = \frac{1}{n} < 1,$

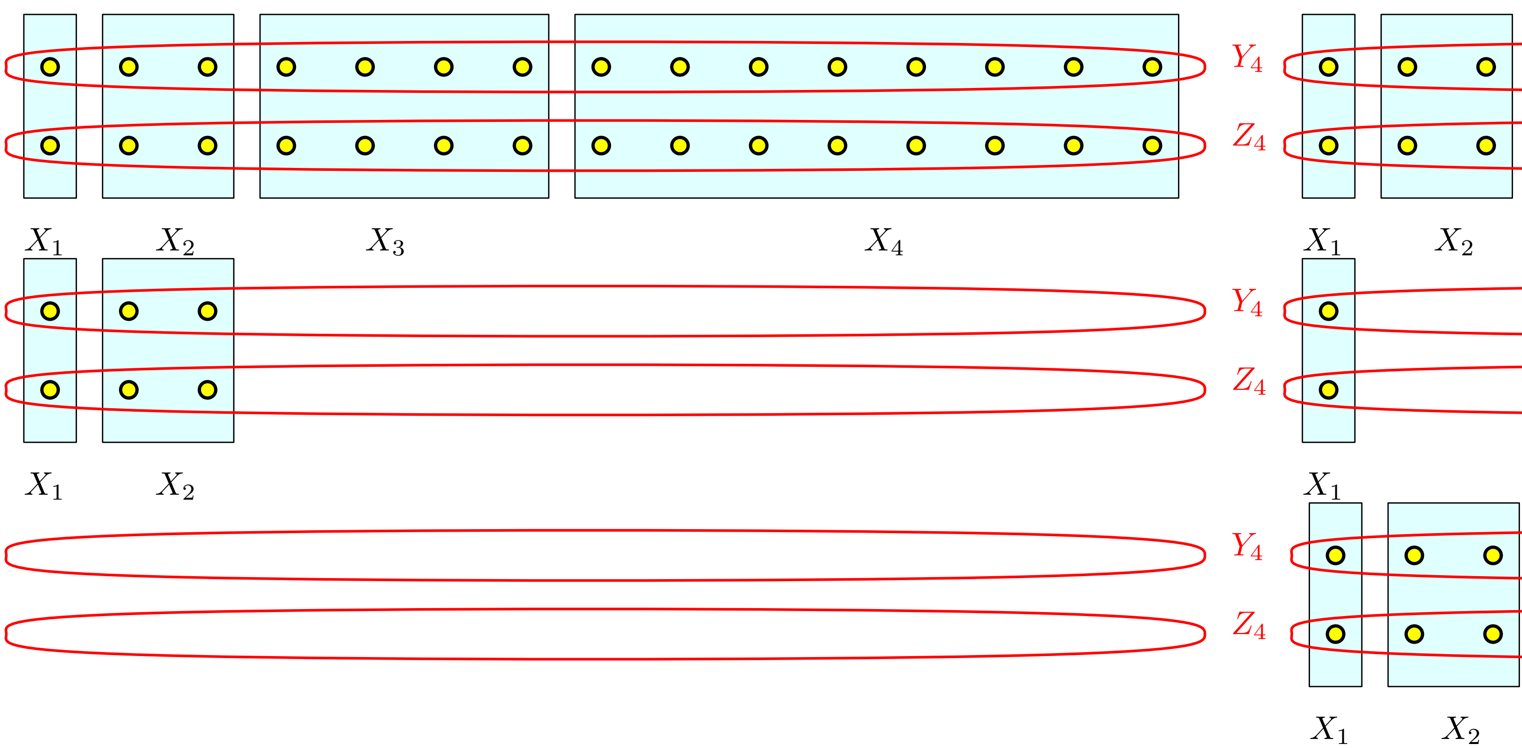
$\implies$ $|U_M| = 0$

$\implies$ Algorithm terminates before reaching Mth iteration. ■

8.2.9 Lower bound

8.2.10 Set cover – Greedy algorithm

8.2.10.1 Lower bound



Lemma 8.2.5. *Let $n = 2^{i+1} - 2$. $\exists$ instance of* ***Set Cover*** *of n elements.*
Optimal cover is by two sets.
GreedySetCover *would use $i = \lfloor \lg n \rfloor$ sets.*
GreedySetCover *is a $\Theta(\log n)$ approximation to* ***SetCover****.*

8.2.11 Just for fun – weighted set cover

8.2.11.1 Weighted set cover

Weighted Set Cover

Instance: $(\mathsf{S}, \mathcal{F}, \rho)$:
S: a set of n elements
$\mathcal{F}$: family subsets of S, s.t. $\bigcup_{X \in \mathcal{F}} X = \mathsf{S}$.
$\rho(\cdot)$: A price function assigning price to each set in $\mathcal{F}$.
Question: The set $\mathcal{X} \subseteq \mathcal{F}$, such that $\mathcal{X}$ covers S. Formally, $\bigcup_{X \in \mathcal{X}} X = \mathsf{S}$, and $\rho(\mathcal{X}) = \sum_{X \in \mathcal{X}} \rho(X)$ is minimized.

(A) **WGreedySetCover**: repeatedly picks set that pays the least cover each element it covers.
(B) $X \in \mathcal{F}$ covered t new elements, then the ***average price*** it pays per element $\beta(X) = \rho(X) / t$.
(C) **WGreedySetCover**: picks the set with the lowest average price.

8.2.12 Weighted set cover – greedy algorithm

8.2.12.1 Analysis

(A) U_i: set of elements not covered in beginning ith iteration.
(B) $U_1 = \mathsf{S}$.
(C) Opt: optimal solution.
(D) ***average optimal cost***: $\beta_i = \rho(\mathrm{Opt}) / |U_i|$,

8.2.13 Weighted set cover – greedy algorithm

8.2.13.1 Analysis – continued

Lemma 8.2.6. *(A) $\beta_1 \le \beta_2 \le \cdots$.*
(B) For $i < j$, we have if $|U_j| > |U_i| / 2$ then $2\beta_i > \beta_j$.

Proof: (A) $\beta_i = \frac{\rho(\mathrm{Opt})}{|U_i|}$: $\rho(\mathrm{Opt})$ is constant and $|U_i|$ can only decrease
(B) $|U_j| > |U_i| / 2 \implies 2 / |U_i| > 1 / |U_j| \implies 2\rho(\mathrm{Opt}) / |U_i| > \rho(\mathrm{Opt}) / |U_j| \implies 2\beta_i > \beta_j$ ■

8.2.14 Weighted set cover – greedy algorithm

8.2.14.1 Analysis – continued

Lemma 8.2.7. *$\beta_i = \rho(\mathrm{Opt}) / |U_i|$: average optimal cost per uncovered element.*
Let $\mathrm{Opt} = \{X_1, \ldots, X_m\}$, and $s_j = |U_i \cap X_j|$.
Then $\exists X_j \in \mathrm{Opt}$ with lower average cost: $\rho(X_j) / s_j \le \beta_i$.

Proof:

$$\min_{j=1}^{m} \frac{\rho(X_j)}{s_j} \leq \frac{\sum_{j=1}^{m} \rho(X_j)}{\sum_{j=1}^{m} s_j} = \frac{\rho(\mathrm{Opt})}{\sum_{j=1}^{m} s_j} \leq \frac{\rho(\mathrm{Opt})}{|U_i|} = \beta_i.$$

Main Point Greedy pays at most β_i per element in round i.

8.2.15 Weighted set cover – greedy algorithm

8.2.15.1 Analysis – continued

Lemma 8.2.8. *k: first iteration $|U_k| \leq n/2$.*
Total price of sets picked in iterations $1 \dots k-1$, is $\leq 2\rho(\mathrm{Opt})$.

Proof: (A) $|U_j| > |U_1|/2$ for $j = 2, \dots, k-1$,
(B) Earlier we showed: if $|U_j| > |U_1|/2$ then $2\beta_1 > \beta_j$.
(C) $\beta_j = \frac{\rho(\mathrm{Opt})}{|U_j|}$ and $|U_1| = n$
$\Rightarrow 2\rho(\mathrm{Opt})/n > \beta_j$ for $j = 1, \dots, k-1$
(D) We showed greedy pays at most β_j per element in round j
$\Rightarrow$ in rounds $j = 1, \dots, k-1$ greedy paid at most twice what opt paid per element.

8.2.16 Weighted set cover – greedy algorithm

8.2.16.1 The result

Theorem 8.2.9. **WGreedySetCover** *computes a $O(\log n)$ approximation to the optimal weighted set cover solution.*

Proof: (A) By Lemma: **WGreedySetCover** paid at most twice the Opt price to cover half the elements.
(B) Now, repeat the argument on the remaining uncovered elements.
(C) After $O(\log n)$ such halving steps, all sets covered.
(D) In each halving step, **WGreedySetCover** paid at most twice the opt cost.

8.3 Clustering

Part I

Clustering

8.3.0.2 Clustering

(A) ***unsupervised learning***: Given examples, partition them into classes of similar examples.
(B) Example: Given webpage X about "The reality dysfunction", find all webpages on this topic (or closely related topics).
(C) Webpage about "All quiet on the western front" should be in the same group as webpage as "Storm of steel".
(D) Hope: All such webpages of interest in same cluster as X, if the clustering is good.

8.3.0.3 Clustering – similarity measure

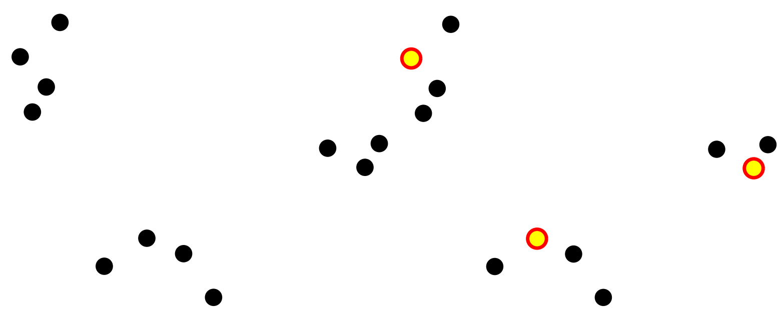
(A) Input: A set of examples (points in high dim).
(B) Example:
    (A) Webpage W: ith coordinate to 1 if the word w_i appears in W.
    We have $10,000$ words care about.
    W interpreted as a point $\in \{0,1\}^{10,000}$.
    (B) Let X be the resulting set of n points in d dimensions.
(C) Need similarity measure.
(D) For example, Euclidean distance between points, where

$$\|p - q\| = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2},$$

where $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$.

8.3.0.4 Clustering – k center clustering

k center clustering problem P: set of n cities, and distances between them.
Build k hospitals, s.t. max dist city from its closest hospital is min.
Example: $k = 3$

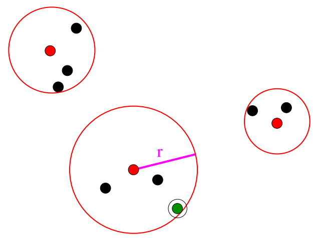


8.3.0.5 Clustering – price

(A) ***price*** of clustering of P by S is

$$\nu(P, S) = \max_{p \in P} \mathbf{d}(p, S)$$

(B) ***k-center*** problem.
    (A) Find $S \subseteq P$ s.t. $|S| = k$ and $\nu(P, S)$ minimized.
    (B) Equivalently, find k smallest discs centered at input points...
    (C) ... cover all the points of P.



8.3.0.6 k Center Clustering

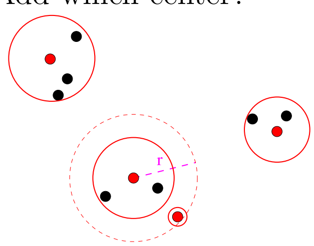
(A) k-center clustering is **NP-Hard**...
(B) ...even to approximate within a factor of (roughly) 1.8.
(C) Formal definition...

k-center clustering

> **Instance**: A set P of n points, a distance function $\mathbf{d}(p, q)$, for $p, q \in P$, satisfying the triangle inequality, and a parameter k.
> **Question:** Find the subset S that realizes
> $r_{opt}(P, k) = \min_{S \subseteq P, |S|=k} \nu(P, S)$,
> where $\nu(P, S) = \max_{p \in P} \mathbf{d}(p, S)$

8.3.0.7 k-center clustering - approximation

(A) Current solution with 3 centers... Add which center?



(B) ...use bottleneck point.
(C) Point furthest away from centers.
(D) Find a new center that better serves this bottleneck point.
(E) ...make it the next center.

8.3.0.8 k-center clustering - approximation algorithm

```
AprxKCenter(P, k)
    P = {p_1, ..., p_n}
    S = {p_1}, u_1 ← p_1
    while |S| < k do
        i ← |S|
        for j = 1...n do
            d_j ← min(d_j, d(p_j, u_i))
        r_{i+1} ← max(d_1, ..., d_n)
        u_{i+1} ← point of P realizing r_i
        S ← S ∪ {u_{i+1}}

    return S
```

8.3.0.9 k-center clustering - approximation algorithm

(A) Running time of **AprxKCenter** is $O(nk)$

(B) r_{i+1}: the (minimum) radius of the i balls centered at $u_1, \ldots, u_i$ covering P.
(C) $\exists p \in P$: $\mathbf{d}(p, \{u_1, \ldots, u_i\}) = r_{i+1}$.
(D) Imagine run **AprxKCenter** one additional iteration.
... so r_{k+1} is well defined.

8.3.1 k-center clustering approximation algorithm

8.3.1.1 Analysis

Lemma 8.3.1. $r_2 \geq \ldots \geq r_k \geq r_{k+1}$.

Proof...

Observation 8.3.2. *The radius of the clustering generated by* **AprxKCenter** *is* r_{k+1}.

8.3.2 k-center clustering approximation algorithm

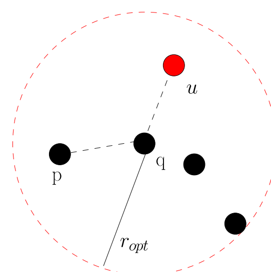
8.3.2.1 Analysis – continued

Lemma 8.3.3. $r_{k+1} \leq 2r_{opt}(P, k)$. $\quad r_{opt}(P, k)$*: radius of the opt with* k *balls.*

Proof:
(A) $D_1, \ldots, D_k$: k discs in opt sol.
(B) S: k centers computed by **AprxKCenter**.

(C) Suppose every disk D_i contains at least one point of S...
(D) Then $\forall p \in P$ distance to S is $\leq 2r_{opt}(P, k)$. That is,

$$\mathbf{d}(p, u) \leq \mathbf{d}(p, q) + \mathbf{d}(q, u) \leq 2r_{opt}$$

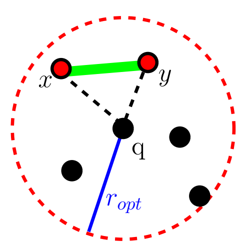


8.3.3 k-center clustering approximation algorithm

8.3.3.1 Analysis – continued

Proof continued
(A) Otherwise, $\exists x, y \in \mathsf{S}$ contained in same ball D_i of Opt.

(B) Let D_i be centered at a point q.
(C) Claim: $\mathbf{d}(x, y) \geq r_{k+1}$.
(D) Suppose $u_\alpha = x$, $u_\beta = y$, $\alpha < \beta$.
(E) $\mathbf{d}(x, y) \geq \mathbf{d}(y, \{u_1, \ldots, u_{\beta-1}\}) \geq r_\beta$



(F) By lemma $r_\beta \geq r_{k+1}$. $\implies$ claim holds
(G) By triangle inequality: $r_{k+1} \leq \mathbf{d}(x, y) \leq \mathbf{d}(x, q) + \mathbf{d}(q, y) \leq 2r_{opt}$. ∎

8.3.3.2 k-center clustering approximation algorithm

Theorem 8.3.4. *One can approximate the k-center clustering up to a factor of two, in time $O(nk)$.*

Proof: **AprxKCenter**: approximation algorithm.

The approximation quality guarantee follows from the above lemma, since the furthest point of P from the k-centers computed is r_{k+1}, which is guaranteed to be at most $2r_{opt}$. ■

8.4 Subset Sum

8.4.0.3 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.

8.4.0.4 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...Mn] - boolean array init to false.
        // b[x] is true if x can be realized
                        // by a subset of X.
    b[0] ← true.
    for i = 1,...,n do
        for j = Mn down to x_i do
            b[j] ← B[j − x_i] ∨ B[j]

    return B[t]
```

8.4.1 Subset Sum

8.4.1.1 Efficient algorithm???

(A) Algorithm solving **Subset Sum** in $O(Mn^2)$.
(B) M might be prohibitly large...
(C) if $M = 2^n \implies$ 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 γ_{opt} one can represent as a subset sum of X which is smaller or equal to t.

8.4.2 Subset Sum

8.4.2.1 2-approximation

Lemma 8.4.1. *(A) (X, t); Given instance of* ***Subset Sum****. $\gamma_{\mathrm{opt}} \le t$: Opt.*
(B) $\implies$ Compute legal subset with sum $\ge \gamma_{\mathrm{opt}}/2$.
(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 $s' + u > t \implies t < s' + u < s' + s' = 2s' \implies s' \ge t/2$.

8.4.3 On the complexity of ε-approximation algorithms

8.4.3.1 Polynomial Time Approximation Schemes

Definition 8.4.2 (PTAS). **PROB**: **Max**imization problem.
$\varepsilon > 0$: approximation parameter.
$\mathcal{A}(I, \varepsilon)$ is a ***polynomial time approximation scheme*** (**PTAS**) for **PROB**:
(A) $\forall I$: $(1-\varepsilon)\left|\mathsf{opt}(I)\right| \le \left|\mathcal{A}(I, \varepsilon)\right| \le \left|\mathsf{opt}(I)\right|$,
(B) $|\mathsf{opt}(I)|$: opt price,
(C) $|\mathcal{A}(I, \varepsilon)|$: price of solution of $\mathcal{A}$.
(D) $\mathcal{A}$ running time polynomial in n for fixed ε.
For minimization problem: $|\mathsf{opt}(I)| \le |\mathcal{A}(I, \varepsilon)| \le (1+\varepsilon)|\mathsf{opt}(I)|$.

8.4.3.2 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 8.4.3 (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.

8.4.3.3 Approximating Subset Sum

Subset Sum Approx

Instance: (X, t, ε): 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 $(1-\varepsilon)\gamma_{\text{opt}} \leq z \leq \gamma_{\text{opt}} \leq t$.

8.4.4 Approximating Subset Sum

8.4.4.1 Looking again at the exact algorithm

```
ExactSubsetSum(S, t)
    n ← |S|
    P_0 ← {0}
    for i = 1...n do
            P_i ← P_{i-1} ∪ (P_{i-1} + x_i)
            Remove from P_i all elements > t

    return largest element in P_n
```

(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.

8.4.4.2 Trim the lists...

```
Trim(L', δ)
    L ← Sort(L')
    L = ⟨y_1 ... y_m⟩
    curr ← y_1
    L_out ← {y_1}
    for i = 2...m do
        if y_i > curr · (1 + δ)
            Append y_i to L_out
            curr ← y_i
    return L_out
```

Definition 8.4.4. For two positive real numbers $z \leq y$, the number y is a δ-approximation to z if $\dfrac{y}{1+\delta} \leq z \leq y$.

Observation 8.4.5. *If $x \in L'$ then there exists a number $y \in L_{out}$ such that $y \leq x \leq y(1+\delta)$, where $L_{out} \leftarrow$ **Trim**(L', δ).*

8.4.4.3 Trim the lists...

```
Trim(L', δ)
    L ← Sort(L')
    L = ⟨y_1 ... y_m⟩
    curr ← y_1
    L_out ← {y_1}
    for i = 2...m do
        if y_i > curr · (1 + δ)
            Append y_i to L_out
            curr ← y_i
    return L_out
```

```
ApproxSubsetSum(S, t)
    // S = {x_1, ..., x_n},
    //      x_1 ≤ x_2 ≤ ... ≤ x_n
    n ← |S|, L_0 ← {0}, δ = ε/2n
    for i = 1...n do
        E_i ← L_{i-1} ∪ (L_{i-1} + x_i)
        L_i ← Trim(E_i, δ)
        Remove from L_i elems > t.

    return largest element in L_n
```

8.4.4.4 Analysis

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

Claim 8.4.6. *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,

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

8.4.4.5 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, $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1}\alpha' + x_i \leq (1+\delta)^{i-1}(\alpha' + x_i) \leq (1+\delta)^i x'$. ■

8.4.4.6 Running time
8.4.4.7 Running time of ApproxSubsetSum

Lemma 8.4.7. *For $x \in [0,1]$, it holds $\exp(x/2) \leq (1+x)$.*

Lemma 8.4.8. *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.

8.4.4.8 Running time of ApproxSubsetSum

Observation 8.4.9. *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 8.4.10. $|L_i| = O\Big((n/\varepsilon^2)\log n\Big)$, *for $i = 1, \ldots, n$.*

8.4.4.9 Running time of ApproxSubsetSum

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

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

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

$$|L_i| \le |L_{i-1}| + O\!\left(\frac{\ln n}{\delta}\right) \le |L_{i-1}| + O\!\left(\frac{n \ln n}{\varepsilon}\right)$$
$$= O\!\left(\frac{n^2 \log n}{\varepsilon}\right).$$

8.4.4.10 Running time of ApproxSubsetSum

Lemma 8.4.11. *The running time of* **ApproxSubsetSum** *is* $O\Bigl(\frac{n^3}{\varepsilon}\log^2 n\Bigr)$.

Proof: (A) Running time of **ApproxSubsetSum** dominated by total length of $L_1, \ldots, L_n$.
(B) Above lemma implies $\sum_i |L_i| = O\!\left(\frac{n^3}{\varepsilon}\log n\right)$.
(C) **Trim** sorts lists. ith iteration R.T. $O(|L_i| \log |L_i|)$.
(D) Overall, R.T. $O(\sum_i |L_i|\,|L_i|) = O\Bigl(\frac{n^3}{\varepsilon}\log^2 n\Bigr)$.

8.4.4.11 ApproxSubsetSum

Theorem 8.4.12. **ApproxSubsetSum** *returns* $u \le t$, *s.t.* $\frac{\gamma_{\text{opt}}}{1+\varepsilon} \le u \le \gamma_{\text{opt}} \le t$,
γ_{opt}*: opt solution.*
Running time is $O\Bigl((n^3/\varepsilon)\log^2 n\Bigr)$.

Proof: (A) Running time from above.
(B) $\gamma_{\text{opt}} \in P_n$: optimal solution.
(C) $\exists z \in L_n$, such that $z \le \mathsf{opt} \le (1+\delta)^n z$
(D) $(1+\delta)^n = (1+\varepsilon/2n)^n \le \exp\Bigl(\frac{\varepsilon}{2}\Bigr) \le 1+\varepsilon$, since $1+x \le e^x$ for $x \ge 0$.
(E) $\gamma_{\text{opt}}/(1+\varepsilon) \le z \le \mathsf{opt} \le t$.