Chapter 5

Approximation Algorithms

An algorithm that returns near-optimal solutions is called an approximation algorithm.

If C^* denotes the optimal solution of an optimization problem, and C the solution obtained by an approximation algorithm for the same problem, then

if $max\{\frac{C}{C^*}, \frac{C^*}{C}\} \leq \rho(n) \quad \forall n$ instances
then $\rho(n)$ is a ratio bound.

We are assuming that all $C's$ are positive.
For maximization problems : $0 < C \leq C^*$,
for minimization problems : $0 < C^* \leq C$.

Note:

1. i) $\rho(n) \geq 1$ since $max\{\frac{C}{C^*}, \frac{C^*}{C}\} \geq 1$
2. ii) $\rho(n) = 1 \Leftrightarrow C = C^*$.

The relative error of the approximation algorithm is $\frac{|C-C^*|}{C^*}$.
An approximation algorithm has a relative error bound of $\epsilon(n)$ if: $\frac{|C-C^*|}{C^*} \leq \epsilon(n)$.

Fact: $\epsilon(n) \leq \rho(n) - 1$

Our goal is to find approximation algorithms that yield very small ration bounds. For some problems, approximation algorithms have been developed that have fixed ratio bounds, independent of n.

5.1 The vertex cover problem

Given a graph $G = (V, E)$ find a subset $V' \subseteq V$ such that if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$, of smallest size.

```
Approx-Vertex-Cover(G)
C ← 0          {C contains vertex cover being constructed}
E' ← E[G]
while E' ≠ 0
   do let (u, v) be an arbitrary edge of E'
      C ← C ∪ {u, v}
      remove from E' every edge incident on either u or v
return C
```

The algorithm has running time $O(E)$.

Theorem 5.1.1 *Approx-Vertex-Cover has a ratio bound of 2.*

Proof: We show that $\frac{|C|}{|C^*|} \leq 2$.

First note that the algorithm returns a vertex cover C since it loops till every edge in $E[G]$ has been covered by some vertex in C.

Let A be the set of edges that are chosen in step 4 above. No two edges in A share an endpoint, since right after picking an edge, we remove from E' all other edges that are incident on its endpoints (see step 6).

So each execution of step 5 adds two new vertices to C, and $|C| = 2|A|$. No edges of A are adjacent. In other words, any cover contains one vertex for each edge in A.

So $|C^*| \geq |A|$ [since C^* is a cover]

Hence, $(|C| = 2|A|) \wedge (|C^*| \geq |A|) \Rightarrow |C^*| \geq \frac{|C|}{2}$

i.e. $\frac{|C|}{|C^*|} \leq 2$ □

5.2 The Traveling-Salesperson Problem

A complete graph $G = (V, E)$ is given that has nonnegative integer costs $C(u, v)$ for the edges $(u, v) \in E$. We want to find a Hamiltonian cycle (or tour) of G with minimum cost. Let $c(A) = \sum_{(u,v)\in A} c(u, v)$ where $A \subseteq E$. Cost c satisfies the triangular inequality if $c(u, w) \leq c(u, v) + c(v, w)$ for all vertices $u, v, w \in V$.

A) The **TSP** with triangular inequality

	Approx-TSP-Tour(G, c)
1	Select a vertex $r \in V[G]$ to be the root
2	Use Prim's algorithm to construct a $MSTT$ using r as root
3	Traverse the tree in preorder to form the list of visited vertices L
4	Return the Hamiltonian cycle H that visits the vertices as specified in L.

□The running time of Approx-TSP-Tour is $\Theta(E) = \Theta(V^2)$ since the input is a complete graph.

The algorithm returns a tour whose cost is not more than twice the cost of an optimal tour.

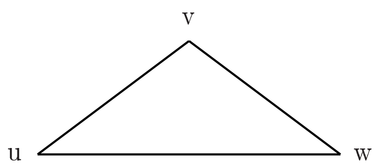
Theorem 5.2.1 Approx-TSP-Tour *is an approximation algorithm with a ratio bound of 2 for the* ***TSP*** *with triangular inequality.*

Proof: H^* denotes an optimal tour. We want to show that $\frac{c(H)}{c(H^*)} \leq 2$, where H is obtained by the approximation algorithm.

Note: $c(T) \leq c(H^*)$ since **MST** T is obtained by deleting an edge from a tour.

A full walk of T, denoted by W, lists the vertices of T when they are first visited and then when they are returned to after visiting the subtree. Thus W traverses every edge of T exactly twice.

So $c(W) = 2c(T)$ therefore $c(W) \leq 2c(H^*)$. So the cost of W is within a factor of 2 of the cost of an optimal tour. W is generally not a tour. By the triangular inequality, vertices can be deleted from W without having the cost increased.



If v is deleted from W between visits to u and w, the result is having to go directly from u to w.

We can thus systematically remove every vertex that has already been visited in W. We end up with a preorder walk of T. Let H be the cycle associated with that walk. It is a Hamiltonian cycle (every vertex is visited once), and it is the cycle obtained by the approximation algorithm.

So $c(H) \leq c(W)$ [H has less vertices than W]

Recall: $c(W) \leq 2c(H^*)$ therefore $\frac{c(H)}{c(H^*)} \leq 2$. □

B) The general **TSP**

Theorem 5.2.2 *If $P \neq NP$ and the ratio bound $\rho \geq 1$, then there exists no polynomial time approximation algorithm with ration bound for the general **TSP**.*

Proof: Proof by contradiction.

Suppose that for some $\rho \geq 1, \exists$ a polynomial time approximation algorithm A with ratio bound ρ. Without loss of generality assume ρ is an integer. We show, that we can use A to solve **Ham-Cycle** in polynomial time.

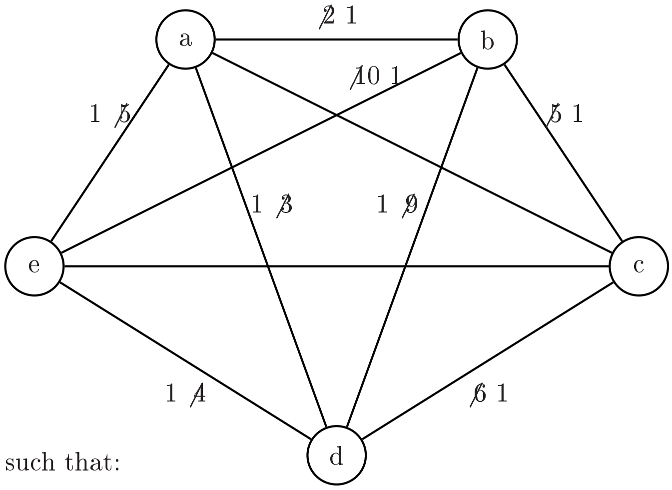
Let $G = (V, E)$ be an instance of **Ham-Cycle**. Convert G into an instance of the **TSP**. Let $G' = (V, E)$ be the complete graph on V: i.e. $E' = \{(u, v) | u, e \in V \text{ and } u \neq v\}$. Assign integer costs to edges of E':

$$c(u, v) = \begin{cases} 1 & \text{if}(u, v) \in E \\ \rho|V| + 1 & \text{otherwise} \end{cases}$$

The conversion from G to G' is in polynomial time in $|V|$ and $|E|$. Consider the **TSP**(G', c). If G has a Hamiltonian circuit H, then (G', c) contains a tour of cost $|V|$ consisting of the edges from H. Recall $c(u, v) = 1$ if $(u, v) \in E$. If G does not contain a Hamiltonian circuit then any tour H' of G' must use some edge not in E. So $c(H') \geq (\rho|V| + 1) + (|V| - 1)$ i.e. $c(H') > \rho|V|$ So a tour on G that is a Hamiltonian circuit costs $|V|$, any other tour costs $> \rho|V|$. Apply A to **TSP** given by (G', c). A returns a tour of cost no more than ρ times the cost of an optimal tour. So if G contains a Hamiltonian circuit, A returns it. If G has no Hamiltonian circuit, then A returns a tour of cost more than $\rho|V|$. So A can be used to solve the HAM-CYCLE in polynomial time. □

So our assumption is incorrect and such an algorithm A cannot exist.

EXAMPLE:



Edges in green are such that:

$c(a,c) = c(e,c) = \rho|V| + 1 = \rho(5) + 1 = 5\rho + 1$ A Hamiltonian cycle of an undirected graph $G = (V, E)$ is a simple cycle that contains each vertex in V. A graph that contains a Hamiltonian cycle is a Hamiltonian graph.

5.3 The Set-Covering Problem

An instance (X, F) of the set-covering problem consists of a finite set X and a family F of subsets of X, such that

$$X = \bigcup_{S \in F} S$$

i.e. every element of X belongs to at least one subset in F. $S \in F$ covers its elements. We are interested in finding a minimum-size subset $C \subseteq F$ whose members cover all of X:

$$X = \bigcup_{S \in C} S$$

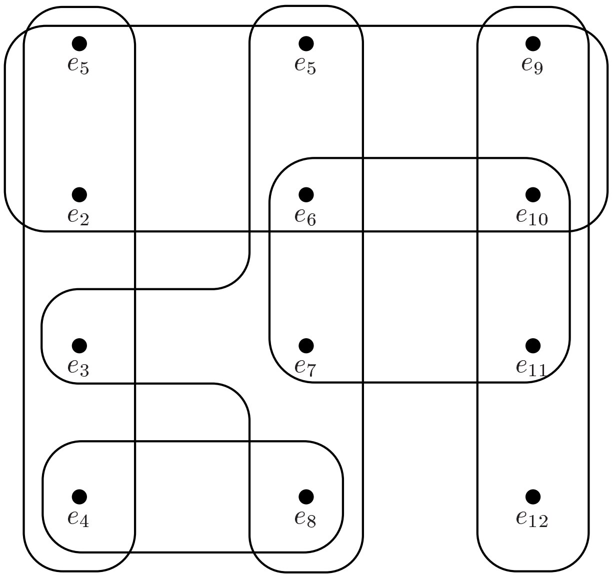
C then covers it elements.
Greedy-set-cover picks, at each iteration, the subset that covers as many uncovered elements as possible.

The algorithm can be implemented to run in polynomial time in $|X|$ and $|F|$. The number of iterations of the loop [line 3-6] is at most $min(|X|, |F|)$. The loop body can be implemented to run in $O(|X||F|)$ time. So the algorithm can run in $O(|X||F|min(|X|, |F|))$.

Greedy-Set-Cover(X, F)

$U \leftarrow X$	*U: set of uncovered elements*
$C \leftarrow \emptyset$	*C: contains cover being constructed*
while $U \neq \emptyset$	
do select $S \in F$ that maximizes $\lvert S \cap U \rvert$	
$U \leftarrow U - S$	*choose subset that contains the maximum number of uncovered elements.*
$C \leftarrow C \cup \{S\}$	
return C	

Example:



An instance (X, F) of the set-covering problem, where X consists of 12 black points and $F = \{S_1, S_2, S_3, S_4, S_5, S_6\}$. A minimum-size set cover is $C = \{S_3, S_4, S_5\}$. The greedy algorithm however produces a cover of size 4 by selecting the sets S_1, S_4, S_5 and S_3 in that order.

5.3.1 Analysis of Greedy-Set-Cover

Theorem 5.3.1 GREEDY-SET-COVER *has a ratio bound* $H(max\{|S| : S \in F\})$ *where*

$$H(d) = \sum_{i=1}^{d} \frac{1}{i}$$

is the d^{th} *harmonic number.*

Proof: Let C^* denote an optimal set cover and S_i the i^{th} subset picked by the algorithm and C the set cover constructed by the algorithm.

Spread the cost of selecting S_i evenly among the elements covered for the first time by S_i.

Each element $x \in X$ is assigned a cost C_x when it is covered for the first time:

$$C_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup ... \cup S_{i-1})} \; [x \text{ is covered by } S_i]$$

In the EXAMPLE:
$c_{e_1} = c_{e_2} = c_{e_5} = c_{e_6} = c_{e_9} = c_{e_10} = \frac{1}{|S_1|} = \frac{1}{6}$
$c_{e_3} = c_{e_7} = c_{e_8} = \frac{1}{|S_4 - S_1|} = \frac{1}{3}$
$c_{e_4} = c_{e_{12}} = \frac{1}{|S_5 - (S_1 \cup S_4)|} = \frac{1}{2}$
$c_{e_4} = \frac{1}{|S_3 - (|S_1 \cup S_4 \cup S_5)|} = 1$ $\square$
If C is returned by the algorithm, then

$$|C| = \sum_{x \in X} c_x$$

is the number of sets in C

$$\Rightarrow |C| \leq \sum_{S \in C^*} \sum_{x \in S} c_x$$

The proof is based on:

$$\sum_{x \in S} c_x \leq H(|S|)$$

which we prove later.

So

$$\begin{aligned} |C| &\leq \sum_{S\in C^*}\sum_{x\in S} c_x \\ &\leq \sum_{S\in C^*} H(|S|) \\ &\leq |C^*|H(max\{|S| : S\in F\}) \\ \Rightarrow \tfrac{|C|}{|C^*|} &\leq H(max\{|S| : S\in F\}) \end{aligned}$$

□

We still have to show that $\sum c_x \leq H(|T|) \quad \forall T \in F$. Let $u_i = |T-(T_1 \cup T_2 ... \cup T_i)|$ for any $T \in F$ and $i = 1, 2, ..., |C|$. u_i is the number of elements in T remaining uncovered after $T_1T_2...T_i$ have been selected by the algorithm. Let $u_0 = |T|$ and k e the least index such that $u_k = 0$, so that every element in T is covered by at least one of the sets $T_1T_2...T_k$. In the EXAMPLE: $T_1 = S_1, T_2 = S_4, T_3 = S_5$ and $T_4 = S_3$ [order in which sets are chosen]. If $T = S_6$ then $u_0 = |S_6| = 2$.
$u_1 = |T - T_1| = |S_6 - S_1 = |\{e_4, e_\}| = 2$
$u_2 = |T - (T_1 \cup T_2)|| = |S_6 - (S_1 \cup S_4)| = |\{e_4\}| = 1$
$u_3 = |T - (T_1 \cup T_2 \cup T_3)| =_6 -(S_1 \cup S_4 \cup S_5)| = |\{e_4\}| = 1$
$u_4 = |T - (T_1 \cup T_2 \cup T_3 \cup T_4)| = |S_6 - (S_1 \cup S_4 \cup S_5 \cup S_3)| = 0$
So $k = 4$.
Note that:
$u_0 - u_1 = 0$ elements of S_6 are covered for the first time by S_1
$u_1 - u_2 = 1$ element of S_6 is covered for the first time by S_4
$u_2 - u_3 = 0$ elements of S_6 are covered for the first time by S_5
$u_3 - u_4 = 1$ element of S_6 is covered for the first time by S_3
So, $u_{i-1} \geq u_i$ and $u_{i-1} - u_i$ elements of T are covered for the first time by T_i, for $i = 1, 2, ..., k$. So

$$\sum_{x\in T} c_x = \sum_{i=1} k(u_{i-1} - u_i)\frac{1}{|T_i - (T_1 \cup T_2 \cup ... \cup T_{i-1})|}$$

Note that: $|T_i - (T_1 \cup T_2 \cup ... \cup T_{i-1}) \geq |T - (T_1 \cup T_2 \cup ... \cup T_{i-1})|$ because the algorithm is greedy and chooses T_i over T. But $u_{i-1} = T - (T_1 \cup T_2 \cup ... \cup T_{i1})|$. So

$$\sum_{x\in T} c_x \leq \sum_{i=1}^{k}(u_{i-1} - u_i)\frac{1}{u_{i-1}}$$

Note: For integers a, b with $a < b$:

$$\begin{aligned} H(b) - H(a) &= \frac{1}{a+1} + \frac{1}{a+2} + ... + \frac{1}{b} \\ H(b) - H(a) &= \sum_{i=a+1}^{b} \frac{1}{i} \\ &\geq \underbrace{\frac{1}{b} + ... + \frac{1}{b}}_{b-a\text{terms}} \end{aligned}$$

$$\Rightarrow H(b) - H(a) \geq (b-a)\frac{1}{b}$$

Back to

$$\begin{aligned}
\sum_{x \in T} c_x &\leq \sum_{i=1}^{k} (H(u_{i-1}) - H(u_i)) \\
&= H(u_o) - H(u_1) + H(u_1) - H(u_2) - ... + H(u_{k-1} - H(u_k) \\
&= H(u_0) - H(u_k)
\end{aligned}$$

But $u_0 = |T|$ and $u_k = 0$

$$\Rightarrow \sum_{x \in T} c_x \leq H(|T|)$$

□

Fact:

$$\text{Since } \sum_{k=1}^{n} \frac{1}{k} \leq \ln n + 1 \text{ we have } \frac{|C|}{|C^*|} \leq \ln |x| + 1$$

□

5.4 The Subset-Sum problem

Given(S, t) where $S = \{x_1, ..., x_n\}, x_i \in Z^+$ and $t \in Z^+$, can we find a subset S' of S, whose elements add up to the target value t?

In the optimization problem, we want to find a subset S' whose sum is as large as possible but not larger than t.

An exponential time algorithm

If $L =< l_1, l_2, ..., l_m >$ is a list of numbers, $L + x =< l_1 + x, l_2 + x, ..., l_m + x >$. Similarly, if $S = \{x_i | x_i \in Z^+, 1 \leq i \leq n\}$
then $S + x = \{x_i + x | x_i \in S \wedge x \in Z^+\}$.

Recall that given 2 sorted lists L and L', we can merge them (via Merge-Lists(L, L') for example) in $O(|L| + |L'|)$ time.

Algorithm Exact-Subset-Sum(S, t) returns a subset S' of S whose sum is as large as possible but not larger than t.

```
EXACT-SUBSET-SUM(S, t)
n ← |S|
L_0 ← < 0 >
for i ← 1 to n
  do L_i ← MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
     remove from L_i every element > t
return largest element in L_n
```

Let P_i be the set of all values that can be obtained by choosing a subset of $\{x_1, x_2, ..., x_i\}$ and adding its members. The subset can be Φ.

Example:

$$\begin{gathered}
S = \{2, 5, 9\} \\
P_1 = \{0, 2\} \\
P_2 = \{0, 2, 5, 7\} \\
P_3 = \{0, 2, 5, 7, 9, 14, 16\}
\end{gathered}$$

It can be shown that:

1. $P_i = P_{i-1} \cup (P_{i-1} + x_i)$, and

2. L_i in Exact-Subset-Sum$S, t)$ is a sorted list containing every element of P_i whose sum is $\leq t$, and

3. Exact-Subset-Sum(S, t) is an exponential time algorithm in general.

1. An approximation scheme for an optimization problem is an algorithm A which takes as input both:

   - the problem instance I, and
   - an error bound ϵ

   and has the performance guarantee:

$$\frac{A(I, \epsilon)}{\text{opt}(I)} \leq 1 + \epsilon$$

   Note: This is a very desirable result.

2. A polynomial approximation scheme is an approximation scheme $\{A_\epsilon\}$ where each algorithm $A_\epsilon$runs in time polynomial in the length of the input instance I.

3. A polynomial approximation scheme is an approximation scheme $\{A_\epsilon\}$ where each algorithm $A_\epsilon$runs in time polynomial in the length of the input instance I and $\frac{1}{\epsilon}$.

Remark: Having established that a certain problem is $\mathcal{NP}$–COMPLETE, the both possible algorithms for that problem would be the existence of a fully polynomial approximation scheme. Very few $\mathcal{NP}$–COMPLETE problems have fully polynomial approximation schemes.

We achieve a fully polynomial time approximation scheme for the subset-sum problem by trimming (removing elements) each list L_i after it is created.

To trim a list L by $\delta, 0 < \delta \leq 1$, means to remove as many elements from L as possible such that L', the resulting list, consists of elements z that ̈represent ̈the removed elements $y \in L$. More precisely every $y \in L$ that is removed, is represented by a $z \in L'$ such that $\frac{y-z}{y} > \delta$ (or equivalently: $y - z \leq \delta y$ and therefore: $(1 - \delta)y \leq z \leq y$)

Each removed $y \in L$ is represented by a $z \in L'$ such that the relative error of z, w.r.t. y, is at most δ.

For Example: Suppose $L =$<10, 11, 12, 15, 20, 21, 22, 23, 24, 29> and $\delta = 0.1$.

$10 \in L'$	(smallest element always in L')
$\frac{11-10}{11} = \frac{1}{11} \leq 0.1$	so drop 11 (it is represented by 10)
$\frac{12-10}{12} = \frac{2}{12} > 0.1$	include 12 in L' (cannot be represented by 10)
$\frac{15-12}{15} = \frac{3}{15} > 0.1$	so $15 \in L'$
$\frac{20-15}{20} > 0.1$	so $20 \in L'$
$\frac{22-20}{22} \leq 0.1$	so $22 \notin L'$
$\frac{23-20}{23} > 0.1$	so $23 \in L'$
$\frac{24-23}{24} \leq 0.1$	so $24 \notin L'$
and finally	
$\frac{29-23}{29} > 0.1$	so $29 \in L'$

$$\Rightarrow L' =< 10, 12, 15, 20, 23, 29 >$$

Procedure TRIM(L,δ) trims $L =< y_1, \ldots <_m>$ in time $\Theta(m)$. It assumes that L is sorted in nondecreasing order.

It returns L', a trimmed and sorted list.

```
TRIM(L, δ)
m ← |L|
L' ←< y_1 >                      { most recent element to join L'}
for i ← 2 to m
   do if  last < (1 − δ)y_i      { last cannot represent y_i }
      then append y_i onto L'
         last ← y_i
return L'
```

We make use of TRIM (L,δ) to construct an approximation scheme.

```
APPROX-SUBSET-SUM(S, t, ε)
n ← |S|
L_0 ←< 0 >
for i ← 1 to n
   do L_i ← MERGE-LISTS(L_{i−1}, L_{i−1} + x_i)
         L_i ← TRIM(L_i, ε/n)
         remove from L_i elements > t
let z be the longest element of L_n
return z
```

EXAMPLE: Let $L =< 104, 102, 201, 101 >$, $t = 308$ and $\epsilon = 0.20$. Note that the optimal answer is $\{104, 102, 101\}$ and $\begin{array}{r} 104 \\ 102 \\ 101 \\ \hline 307 \end{array}$

APPROX-SUBSET-SUM (S,t,ϵ) yields 302 which is $\frac{307-302}{307} = 0.016\%$ from the optimum.

EXAMPLE: $t = 308$, $\epsilon = 0.2$, $L =<104, 102, 201, 101>$. Note $L_0 \le 0$. Trimming parameter: $\delta = \frac{\epsilon}{4} = 0.05$

i	$L_i \leftarrow$ MERGE-LISTS	$L_i \leftarrow$ TRIM	after removal
1	$L_1 =<0, 104>$	$L_1 \leftarrow<0, 104>$	$L_1 \leftarrow<0, 104>$
2	$<0, 102, 104, 206>$	$<0, 102, 206>$, because $102 \not< (1 - 0.05) \cdot 104$	$<0, 102, 206>$
3	merge $<0, 102, 206>$ and $<201, 303, 407> \Rightarrow$ $<0, 102, 201, 206, 303, 407>$	206 is repr. by 201, so $<0, 102, 201, 303, 407>$	drop 407 since $407 > 308$, so $<0, 102, 201, 303>$
4	merge $<0, 102, 201, 303>$ and $<101, 203, 302, 404>$ $\Rightarrow$ $<0, 101, 102, 201, 203, 302, 303, 404>$	102 is repr. by 101 203 is repr. by 302 303 is repr. by 302 so $<0, 101, 201, 302, 404>$	remove 404 since $404 > 308$, so $<0, 101, 201, 302>$

Hence $z = 302$, while the optimal solution is, as already shown above, equal to $101 + 102 + 104 = 307$.

Theorem 5.4.1 APPROX-SUBSET-SUM *is a fully polynomial time approximation scheme for the subset-sum problem.*

Proof: We have to show that:

1. Approx-Subset-Sum is an approximation scheme
2. Approx-Subset-Sum is fully polynomial time

1. We have to show that the scheme is an an approximation algorithm with relative error bound ϵ. In other words, we have to show that $\frac{C^*-C}{C^*} \leq \epsilon$ or $C^*(1-\epsilon) \leq C$.

   Trimming L_i and removing from it ecery element greater than t is done in such a way that each element of L_i is the sum of elements of some subset of S. In particular, z, returned at the end of Approx-Subset-Sum is the sum of some subset of S. We still have to show that $z = C$ is such that $C^*(1-\epsilon) \leq z$.

   Note that if y represents y', then $\frac{y'-y}{y'} \leq \delta = \frac{\epsilon}{n}$.

   In other words, the relative error between y and y' is at most $\frac{\epsilon}{n}$.

   It can be shown, by induction on i, that for every element $y \in P_i$ where $y \leq t$, there exists a $z \in L_i$ such that:

   $$(1 - \frac{\epsilon}{n})^i y \leq z \leq y$$

   In particular, if $y^* \in P_n$ denotes an optimal solution to the subset-sum problem, then there exists a $z \in L_n$ such that:

   $$(1 - \frac{\epsilon}{n})^n y^* \leq z \leq y^*$$

   and the largest possible z is returned by Approx-Subset-Sum.

   Consider $\frac{d}{d_n}(1-\frac{\epsilon}{n})^n$.

   $$\begin{aligned} \frac{d}{d_n}(1 - \frac{\epsilon}{n})^n &= n(1-\frac{\epsilon}{n})^{n-1}\frac{d}{d_n}(1-\frac{\epsilon}{n}) \\ &= n(1-\frac{\epsilon}{n})^{n-1}(-1)(-\frac{\epsilon}{n^2}) \\ &= n(1-\frac{\epsilon}{n})^{n-1}(\frac{\epsilon}{n^2}) \\ &= \frac{\epsilon}{n}(1-\frac{\epsilon}{n})^{n-1} \end{aligned}$$

   $\frac{d}{d_n}(1-\frac{\epsilon}{n})^n > 0$ so the function

   $$q(n) = (1 - \frac{\epsilon}{n})^n$$

   increases with n.

   So

   $$\begin{aligned} n > 1 &\Rightarrow q(n) > q(1) \\ &\Rightarrow (1-\frac{\epsilon}{n})^n > 1-\epsilon \end{aligned}$$

   So $(1-\epsilon)y^* < (1-\frac{\epsilon}{n})^n y^* \leq z$ But recall: $(1-\frac{\epsilon}{n})^n y^* \leq z$ therefore $(1-\frac{\epsilon}{n})y^* \leq z$

   In other words $\frac{y^*-z}{y^*} \leq \epsilon$
   therefore Approx-Subset-Sum is an approximation scheme.

2. It can be shown the procedure is polynomial in the length of the $L_i's$.

Bibliography

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, 1990.