第8章2 近似算法

第8章2 近似算法

迄今为止，所有的NP完全问题都还没有多项式时间算法。对于这类问题，通常可采取以下几种解题策略。

(1) 只对问题的特殊实例求解

(2) 用动态规划法或分支限界法求解

(3) 用概率算法求解

(4) 只求近似解

(5) 用启发式方法求解

本章主要讨论解NP完全问题的**近似算法**。

近似算法的性能

若一个最优化问题的最优值为c*，求解该问题的一个近似算法求得的近似最优解相应的目标函数值为c，则将该**近似算法的性能比**定义为 $\eta = \max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\}$。

在通常情况下，该性能比是问题输入规模n的一个函数ρ(n)（Ratio Bound），即 $\max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\} \le \rho(n)$。

近似算法的**相对误差定义**为 $\lambda = \left|\frac{c - c^*}{c^*}\right|$。若对问题的输入规模n，有一函数ε(n)使得 $\left|\frac{c - c^*}{c^*}\right| \le \varepsilon(n)$，则称ε(n)为该近似算法的**相对误差界**。

近似算法的性能

- 有许多问题的近似算法具有固定的性能比或相对误差界，即ρ(n)或ε(n)是不随n的变化而变化的。在这种情况下，用ρ和ε来记性能比和相对误差界，表示它们不依赖于n；
- 还有许多问题没有固定性能比的多项式时间近似算法，其性能比只能随着输入规模n的增长而增大；
- 对有些NP完全问题，可以找到这样的近似算法，其性能比可以通过增加计算量来改进，也就是说，在计算量和解的精确度之间取一个折中。

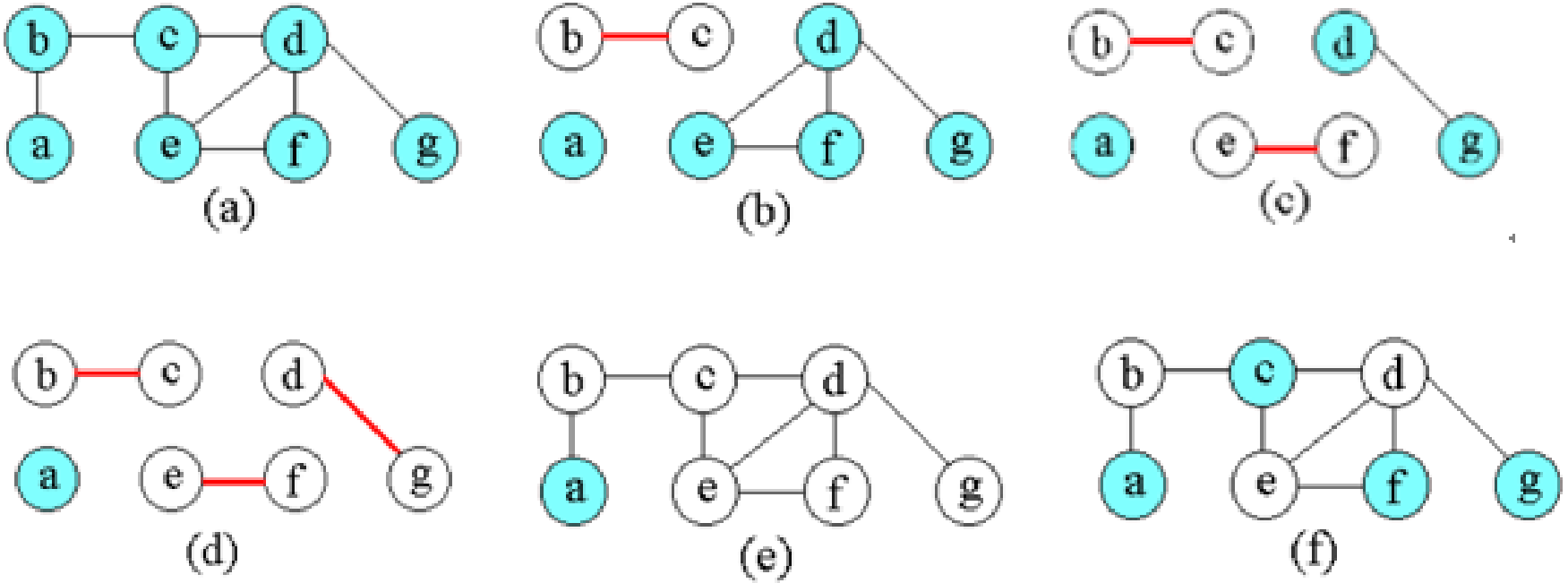
顶点覆盖问题的近似算法

问题描述：无向图G=(V,E)的顶点覆盖是它的顶点集V的一个子集V'⊆V，使得若(u,v)是G的一条边，则v∈V'或u∈V'。顶点覆盖V'的大小是它所包含的顶点个数|V'|。

```
VertexSet approxVertexCover ( Graph g ){
    cset=∅;
    e1=g.e;
    while (e1 != ∅)
        从e1中任取一条边(u,v);
        cset=cset∪{u,v};
        从e1中删去与u和v相关联的所有边;
    return cset;
}
```

cset用来存储顶点覆盖中的各顶点。初始为空，不断从边集e1中选取一边(u,v)，将边的端点加入cset中，并将e1中已被u和v覆盖的边删去，直至cset已覆盖所有边，即e1为空。

顶点覆盖问题的近似算法



图(a)～(e)说明了算法的运行过程及结果。(e)表示算法产生的近似最优顶点覆盖cset，它由顶点b c d e f g所组成。(f)是图G的一个最小顶点覆盖，它只含有3个顶点：b d e。

近似算法性能分析

- 每条边扫描一次，时间复杂度为O(|E|)
- Ratio Bound为2。证明如下：
  - 令E’为选中的边集，若(u,v)∈ E’，则与其相邻的边都被删除，因此E’中无相邻边;
  - 每次选一条边，则每次有两个顶点加入解集A，|A|=2|E’|;
  - 设OPT为最优解，由于E’中无邻接边，OPT至少包含E’中每条边的一个顶点，故|E’|≤|OPT|;
  - 故|A|=2|E’|≤2|OPT|，从而得性能比 |A|/|OPT|≤2。

旅行售货员问题近似算法

问题描述：给定一个完全无向图G=(V,E)，其每一边(u,v)∈E有一非负整数费用c(u,v)。要找出G的最小费用哈密顿回路。

旅行售货员问题的一些**特殊性质**：

比如，费用函数c往往具有**三角不等式性质**，即对任意的3个顶点u,v,w∈V，有：c(u,w)≤c(u,v)+c(v,w)。当图G中的顶点是平面上的点，任意2顶点间的费用是这2点间的欧氏距离时，费用函数c就具有三角不等式性质。

具有三角不等式性质的旅行售货员问题

对于给定的无向图G，可以利用找**图G的最小生成树**算法设计找近似最优的旅行售货员回路的算法。

void **approxTSP** (Graph g){

1) 选择g的任一顶点r；

2) 用Prim算法找出带权图g的一棵以r为根的最小生成树T；

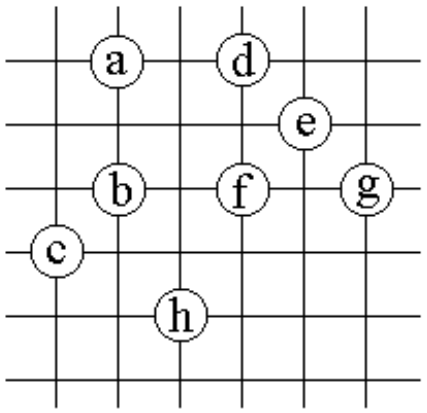
3) 前序遍历树T得到顶点表L；

4) 将r加到表L的末尾，按表L中顶点次序组成回路H，作为计算结果返回；

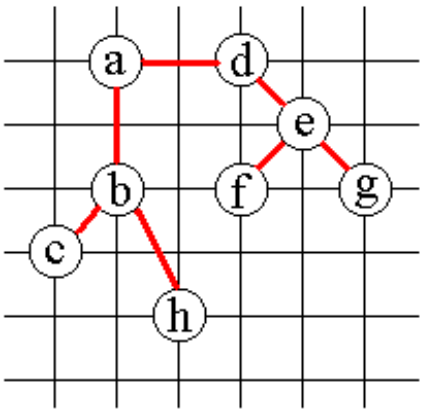
}

第二步的复杂度为$O(|V|^2)$, 第三四步的复杂度为$O(|V|)$, 因为最小生成树恰有$|V|-1$条边

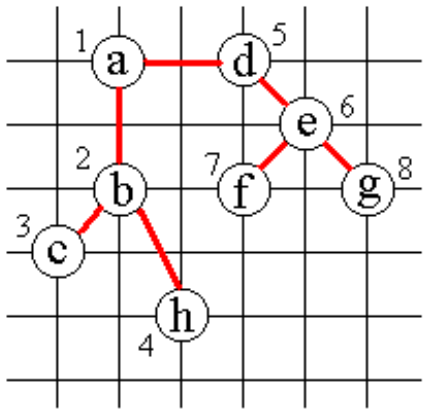
具有三角不等式性质的旅行售货员问题举例



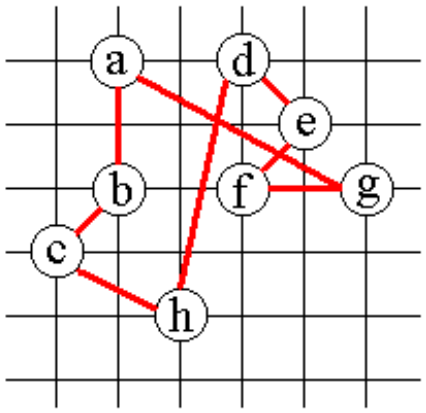
(a)



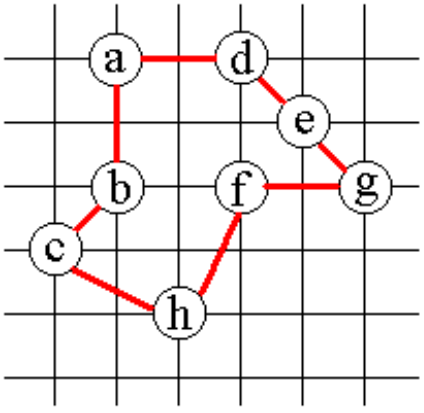
(b)



(c)



(d)



(e)

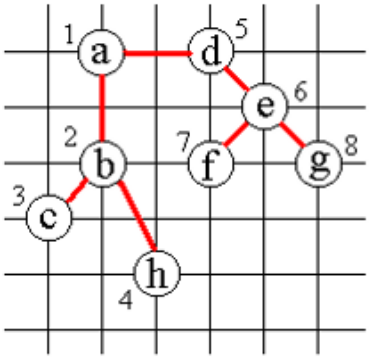
(b) 表示找到的最小生成树T；

(c) 表示对T作前序遍历的次序；

(d) 表示前序遍历顶点表L产生的哈密顿回路H；

(e) 是G的一个最小费用旅行售货员回路。

近似算法性能分析



- 对于最优解OPT，删除一条边可得生成树，且该树的代价一定不低于最小生成树T，$|T| \leqslant |OPT|$.
- 设W是对树T的完全遍历。则W经过T的每条边恰好两次，从而有$|W|=2|T|$。对图中的树，完全遍历W=abcbhbadefegeda；
- L是删除W中某些重复访问的顶点得到的结果，例如uvw中删去v，由三角不等式，用uw取代uv和vw会得到更小的代价；
- 因此，$|L| \leq |W|=2|T| \leqslant 2|OPT|$。

当费用函数满足三角不等式时，算法找出的旅行售货员回路的费用不会超过最优旅行售货员回路费用的2倍。

一般的旅行售货员问题

在费用函数不一定满足三角不等式的一般情况下，不存在具有常数性能比的解TSP问题的多项式时间近似算法，除非**P=NP**。

换句话说，若$P \neq NP$，则对任意常数$\rho>1$，不存在性能比为ρ的解旅行售货员问题的多项式时间近似算法。

证明：假设存在一个解旅行售货员问题的多项式时间近似算法A，其性能比为$\rho>1$。下面说明可以用A解哈密顿回路问题：判断图G=（V,E）是否存在一条哈密顿回路。

一般的旅行售货员问题

设G1=(V, E1)为顶点集V上的一个完全图，E1中每一条边的费用c(u,v)定义为：

$$c(u,v) = \begin{cases} 1, & (u,v) \in E \\ \rho|V| + 1, & (u,v) \in E1 - E \end{cases}$$

如此定义的G1和c可根据G关于|V|和|E|的多项式时间构造。

- 若G有哈密顿回路，则c赋给其每边的费用均为1，G1含有一个费用为|V|的旅行回路；
- 若G不存在哈密顿回路，则G1必用到了不在E中的边，因此(ρ|V|+1)+(|V|-1|)>ρ|V|。

一般的旅行售货员问题

- 用近似算法A解旅行售货员问题(G1,c)，则求出的近似解c(H)≤ρc(H*)，H*为最优解
  - 当G有哈密顿回路时，c(H*)=|V|，由近似算法找到的旅行回路H费用为c(H) ≤ρc(H*)=ρ|V|，可知H中每一条边均属于E，故H是G的一条哈密顿回路；
  - 反之，若近似算法找出的旅行回路c(H)>ρ|V|，则ρ|V|<c(H)≤ρc(H*)，于是c(H*)>|V|，即旅行售货员最优回路H*费用>|V|，故不存在哈密顿回路；
- 用近似算法A求出H之后，只需要再判断其费用是否为|V|，即可判断是否存在一条哈密顿回路；
- A可在多项式时间内完成，在P≠NP下不可能。

集合覆盖问题的近似算法

问题描述: 集合覆盖问题的一个实例〈X,F〉由一个有限集X及X的一个子集族F组成。子集族F覆盖了有限集X。也就是说X中每一元素至少属于F中的一个子集，即$X = \bigcup_{S \in F} S$。对于F中的一个子集$C \subseteq F$，若C中的X的子集覆盖了X，即$X = \bigcup_{S \in C} S$，则称C覆盖了X。集合覆盖问题就是要找出F中覆盖X的最小子集 C^*，使得

$$|C^*| = \min\{|C| \mid C \subseteq F \text{且} C \text{覆盖} X\}$$

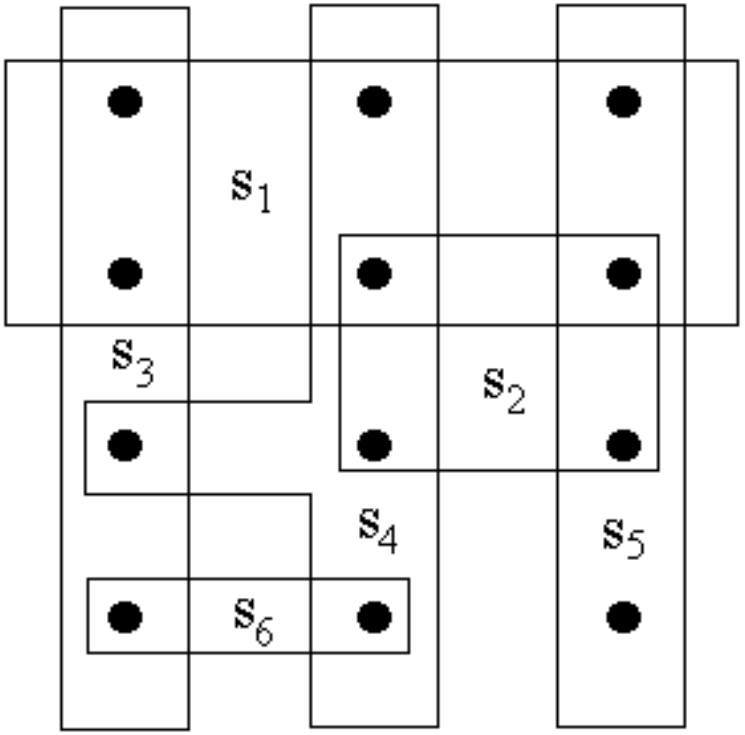
集合覆盖问题的近似算法

集合覆盖问题举例：

用12个黑点表示集合X。

F={S1,S2,S3,S4,S5,S6}，如图所示。

容易看出，对于这个例子，最小集合覆盖为：C={S3,S4,S5}。



集合覆盖问题的近似算法

集合覆盖问题近似算法——贪心算法

```
Set greedySetCover (X, F){
    U=X;
    C=∅;
    while (U !=∅)
        选择F中使|S∩U|最大的子集S;
        U=U-S;
        C=C∪{S};
    return C;
}
```

算法的循环体最多执行min{|X|，|F|}次。而循环体内的计算显然可在O(|X||F|)时间内完成。因此，算法的计算时间为O(|X||F|min{|X|，|F|})。由此即知，该算法是一个多项式时间算法。

集合覆盖问题

性能比分析：

在算法将Si加入子集族C时，给Si一个费用1，并平摊到Si-$U_{j\le i-1}$Sj中的元素上。

算法终止时得到子集族C，其总费用为|C|=$\sum_{x\in X} C_x$。

C*是X的最优覆盖，有 |C|=$\sum_{x\in X} C_x \le \sum_{S\in C*} \sum_{x\in S} C_x$

又有$\sum_{x\in S} C_x$≤H(|S|), 其中H(|S|) = $\sum_{i=1}^{|S|} \frac{1}{i}$， 可得

|C|≤ $\sum_{S\in C*}$ H(|S|) ≤|C*| |H($max_{S\in F}${|S|})，于是ratio bound为$\frac{|C|}{|C*|} \le$ H($max_{S\in F}${|S|})

集合覆盖问题

$\sum_{x\in S} C_x \le H(|S|)$

证明：设$u_i=|S-U_{j\le i} Sj|$表示算法选择S1,…,Si之后，S中尚存的未被覆盖的元素个数，其中u0=|S|，设k为数列u0，u1,…中第一个等于0的下标，则S被S1…Sk所覆盖。

S中有$u_{i-1}-u_i$个元素被Si第一次覆盖，于是有

$$\sum_{x\in S} C_x = \sum_{i=1}^{k} \frac{u_{i-1}-u_i}{|Si-U_{j\le i-1} S_j|} \quad (1)$$

根据贪心算法性质，S所覆盖的新元素不会比Si多，否则算法选择S，于是

$$|Si - U_{j\le i-1} S_j| \ge |S - U_{j\le i-1} S_j| = u_{i-1} \quad (2)$$

集合覆盖问题

$$\sum_{x\in S} C_x = \sum_{i=1}^{k} \frac{u_{i-1}-u_i}{|Si-\mathrm{U}_{j\le i-1}\,\mathrm{S_j}|} \quad (1)$$

$$|Si - \mathrm{U}_{j\le i-1} S_j| \ge |S - \mathrm{U}_{j\le i-1} S_j| = u_{i-1} \quad (2)$$

- 由(1)、(2)可得

$$\sum_{x\in S} C_x = \sum_{i=1}^{k} \frac{u_{i-1}-u_i}{|Si-\mathrm{U}_{j\le i-1}\,\mathrm{S_j}|} \le \sum_{i=1}^{k} \frac{u_{i-1}-u_i}{u_{i-1}}$$

任一正整数a < b，有H(b)-H(a) = $\sum_{i=a+1}^{b} \frac{1}{i} \ge (b-a)/b$，得

$$\sum_{x\in S} C_x \le \sum_{i=1}^{k} \frac{u_{i-1}-u_i}{u_{i-1}}$$

$$\leqslant \sum_{i=1}^{k} [H(u_{i-1}) - H(u_i)]$$

$$= H(u_0) - H(u_k)$$

$$= H(u_0) - H(0)$$

$$= H(|S|)$$

子集和问题的近似算法

问题描述：设子集和问题的一个实例为〈S,t〉。其中，$S=\{x_1, x_2, \ldots, x_n\}$是一个正整数的集合，t是一个正整数。子集和问题判定是否存在S的一个子集S1，使得$\sum_{x \in S1} x = t$。

子集和问题的指数时间算法

```
int exactSubsetSum (S, t){
    int n=|S|;
    L[0]={0};
    for (int i=1; i<=n; i++)
        L[i]=mergeLists(L[i-1],L[i-1]+S[i]);
        删去L[i]中超过t的元素;
    return max(L[n]);
}
```

将集合L[i-1]中每个元素加上S[i]

算法以集合S={x_1, x_2, …, x_n}和目标值t作为输入。算法中用到将2个有序表L1和L2合并成为一个新的有序表的算法mergeLists(L1, L2)。

子集和问题的指数时间算法

- 用P_i表示$\{x_1 \ldots x_i\}$的所有可能的子集和, 有P0={0}, $P_i = P_{i-1} \cup (P_{i-1}+x_i)$
  - 例：S={1,4,5}, P0={0}, P1={0,1}, P2={0,1,4,5}, P3={0,1,4,5,6,9,10}
- 算法得到的表L[i]是一个包含了P_i中所有不超过t的元素的有序表
  - $|P_i|=2^i$，最坏情况下$|L(i)|=|P_i|=2^i$

子集和问题的完全多项式时间近似格式

基于算法exactSubsetSum，通过对表L[i]作适当的修整建立一个子集和问题的完全多项式时间近似格式。

在对表L[i]进行修整时，用到一个修整参数δ，$0 < \delta < 1$。用参数δ修整一个表L是指从L中删去尽可能多的元素，使得每一个从L中删去的元素y，都有一个修整后的表L1中的元素z满足$(1-\delta)y \leqslant z \leqslant y$。可以将z看作是被删去元素y在修整后的新表L1中的代表。

例：若δ=0.1，且L=〈10,11,12,15,20,21,22,23,24,29〉，则用δ对L进行修整后得到L1=〈10,12,15,20,23,29〉。其中被删去的数11由10来代表，21和22由20来代表，24由23来代表。

子集和问题的完全多项式时间近似格式

子集和问题近似格式

```
int approxSubsetSum(S, t, ε){
    n=|S|;
    L[0]=〈0〉;
    for (int i=1; i<=n; i++)
        L[i]=Merge-Lists(L[i-1], L[i-1]+S[i]);
        L[i]=Trim(L[i], ε/n);
        删去L[i]中超过t的元素;
    return max(L[n]);
}
```

对有序表L修整算法

```
List trim(L, δ){
    int m=|L|;
    L1=〈L[1]〉;
    int last=L[1];
    for (int i=2; i<=m; i++) {
        if (last<(1-δ)*L[i]) //无法代表
            将L[i]加入L1的尾部;
            last=L[i];
    return L1;
}
```

子集和问题的完全多项式时间近似格式

对于S={104,102,201,101}，t=308，Ɛ=0.2，修整参数为0.2/4=0.05，每次经过合并、修整、删除三个阶段。

- L[1]={0,104}, L[1]={0,104}, L[1]={0,104}
- L[2]={0,102,104,206}, L[2]={0,102,206}, L[2]={0,102,206}

最终，算法返回近似解302，该例的最优解104+102+101=307。

近似算法性能分析

- 算法计算出的近似解是S的子集和，其关于最优解的相对误差不超过预先设定的ε。
- 算法是子集和问题的一个多项式时间近似，即复杂度是关于n和$1/\varepsilon$的多项式。
  - 对于Pi中任一不超过t的元素y，L[i]中有一x使得$(1-\varepsilon/n)^i y \leqslant x \leqslant y$，对于最优值c*存在于Pn，因此L[n]中有一x使得$(1-\varepsilon/n)^n c^* \leq x \leq c^*$。
  - 由于算法返回L[n]中最大元素z，$x \leq z \leq c^*$，于是$(1-\varepsilon/n)^n c^* \leq z \leq c^*$，利用$(1-\varepsilon) \leq (1-\varepsilon/n)^n$，得到$(1-\varepsilon)c^* \leq z \leq c^*$，进而$(c^*-z)/c^* = 1-z/c^* \leq \varepsilon$。

近似算法性能分析

算法Trim的时间复杂度为O(m)，每次对有序表L[i]做合并、修整、删除的计算时间为O(|L[i]|)，于是整个算法计算时间不超过O(n|L[n]|)。

对L[i]修整后，元素相对间距满足a/b>1/(1-Ɛ/n)，而表中最大元素不超过t，于是元素个数不超过

$$\log_{\frac{1}{1-\frac{\varepsilon}{n}}} t = \frac{lnt}{\ln\frac{1}{1-\frac{\varepsilon}{n}}} = \frac{lnt}{-\ln(1-\frac{\varepsilon}{n})} \leq \frac{t}{\varepsilon/n} = \frac{nt}{\varepsilon}$$

于是整个算法复杂度为$O(n^2t/Ɛ) = O(n^2/Ɛ)$，因t为常数。