第10章 近似算法

1

第10章 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**。

2

10.1 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)，即$\max\left\{\frac{c}{c^*},\frac{c^*}{c}\right\}\leqslant\rho(n)$。

该**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$\left|\frac{c-c^*}{c^*}\right|\leqslant\varepsilon(n)$，则称ε(n)为该**近似算法的相对误差界**。近似算法的性能比ρ(n)与相对误差界ε(n)之间显然有如下关系：

ε(n) ≤ ρ(n)-1。

3

10.2 顶点覆盖问题的近似算法

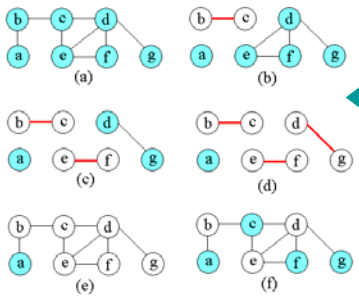
问题描述：无向图G=(V,E)的顶点覆盖是它的顶点集V的一个子集V'⊆V，使得若(u,v)是G的一条边，则v∈V'或u∈V'。顶点覆盖V'的大小是它所包含的顶点个数|V'|。

```
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 c
}
```

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。

4

10.2 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和e。

算法approxVertexCover的性能比为2。

5

10.3 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中的顶点就是平面上的点，任意2顶点间的费用就是这2点间的欧氏距离时，费用函数c就具有三角不等式性质。

6

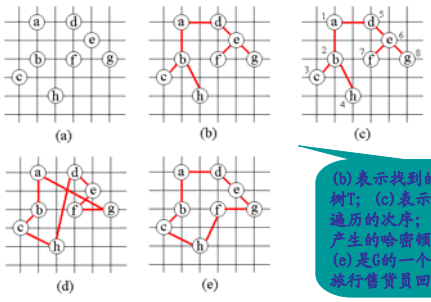
10.3.1 具有三角不等式性质的旅行售货员问题

对于给定的无向图G，可以利用找**图G的最小生成树**的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得到的顶点表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;
}
```

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。

10.3.1 具有三角不等式性质的旅行售货员问题举例



(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。

10.3.2 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。

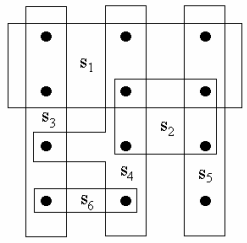
10.4 集合覆盖问题的近似算法

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

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

10.4 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,}。

10.4 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。

10.5 子集合问题的近似算法

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，…，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 $\sum_{x\in S1} x = t$ 。

10.5.1 子集合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]);
}
```

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)。

10.5.2 子集合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＜δ＜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-δ)y≤z≤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。

举例：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。

10.5.2 子集合问题的完全多项式时间近似格式

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]) {
    将L[i]加入表L1的尾部;
    last=L[i];
    }
  return L1;
}
```

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]);
}
```