Lecture 10 近似算法

绳伟光

上海交通大学微纳电子学系

2021-11-11

提纲

提纲

近似算法的基本思想

- 很多实际应用中的问题都是 NP-完全问题
- NP-完全问题的多项式算法是难以得到的 (NP 的内容后续讲)
- 求解 NP-完全问题的方法
  - 如果问题的输入很小, 可以使用指数级算法圆满地解决该问题
  - 如果问题的输入规模较大，可以利用现代优化算法在多项式时间内**高概率**地精确求解
  - 对于输入规模较大的问题，也可使用近似算法求解问题的**近似优化解**
- 什么是近似算法：
  - 能够在多项式时间内给出一个优化问题的近似优化解的算法
  - 近似算法主要解决优化问题，不但可以近似求解 NP-完全问题，还可以近似求解复杂度较高的 P 问题

由于 NP-完全问题的广泛性和计算规模的爆炸性增长，近似算法变得日益重要!

近似算法的性能分析

- 近似算法的时间 (空间) 复杂度：分析的目标和方法与传统算法相同
- **近似度分析**是近似算法特有的，主要用于刻画近似算法给出的近似解相比于问题优化解的优劣程度
- 近似算法求解优化问题
  - 问题的每一个可能的解都具有一个正的代价
  - 问题的优化解可能具有最大或最小代价
  - 我们希望寻找问题的一个近似优化解
- 近似度衡量指标：
  - 近似比
  - 相对误差界
  - $(1+\varepsilon)$-近似

近似比

近似比 (Ratio Bound)

设 A 是一个优化问题的近似算法，设 A 的代价为 C，该问题优化解的代价为 C^*，n 为输入大小，则 A 具有 Ratio Bound $\rho(n)$，如果：$\max(\frac{C}{C^*}, \frac{C^*}{C}) \leq \rho(n)$

- 对于最大化问题，$\max(\frac{C}{C^*}, \frac{C^*}{C}) = \frac{C^*}{C}$
- 对于最小化问题，$\max(\frac{C}{C^*}, \frac{C^*}{C}) = \frac{C}{C^*}$
- Ratio Bound 不会小于 1，Ratio Bound 越大，近似解越坏

相对误差界

相对误差

对于任意输入，相对误差定义为 $|C - C^*|/C^*$，其中 C 是近似算法所产生的近似解的代价，C^* 是问题实例优化解的代价

相对误差界

一个近似算法的相对误差界定义为 $\varepsilon(n)$，如果算法在规模为 n 的任意实例上的相对误差满足 $|C - C^*|/C^* \leq \varepsilon(n)$，其中 $\varepsilon(n)$ 是 n 的函数

- 相对误差界总是非负的
- 对某些近似算法来说，随着运行时间的增加，近似比和相对误差会减小
- $\varepsilon(n) \leq \rho(n) - 1$(请自行证明)

$(1+\varepsilon)$ 近似及近似模式

$(1+\varepsilon)$ 近似算法

优化问题的一个近似算法 A，如果其相对误差界为 $\varepsilon(n)$，则称之为 $(1+\varepsilon)$ 近似算法。$(1+\varepsilon)$ 近似算法既指明了算法的相对误差界又指明了算法的近似比

近似模式

如果优化问题的近似算法 $A(I,\varepsilon)$ 在输入的问题实例 I 和相对误差界 ε 上可输出相对误差不超过 ε 的近似解，则称该近似算法为一个近似模式。近似模式表示了一族近似算法，其中每个算法的近似度各不相同，时间复杂度也不同

近似模式

多项式时间近似模式

如果近似模式 $A(I, \varepsilon)$ 的时间复杂度是 I 的多项式，则称之为多项式时间近似模式

完全多项式时间近似模式

如果近似模式 $A(I, \varepsilon)$ 的时间复杂度既是 I 的多项式，又是 ε 的多项式，则称之为完全多项式时间近似模式

- 运行时间为 $O(n^{2/\varepsilon})$ 的近似算法为多项式时间近似模式，随 ε 的常数倍减小运行时间可能会按指数增长
- 运行时间为 $O((1/\varepsilon)^2 n^3)$ 的近似算法为完全多项式时间近似模式，随 ε 的常数倍减小运行时间只会常数倍增加

不是所有问题均存在近似算法

不是所有问题都存在近似算法！

此论点证明比较复杂，记住结论即可！

提纲

顶点覆盖问题简介

顶点覆盖问题

给定无向图 $G = (V, E)$，它的一个顶点覆盖是一个子集 $V' \subseteq V$，使得如果 $(u, v) \in E$，则 $u \in V'$，或者 $v \in V'$，或者 $\{u, v\} \in V'$ 成立

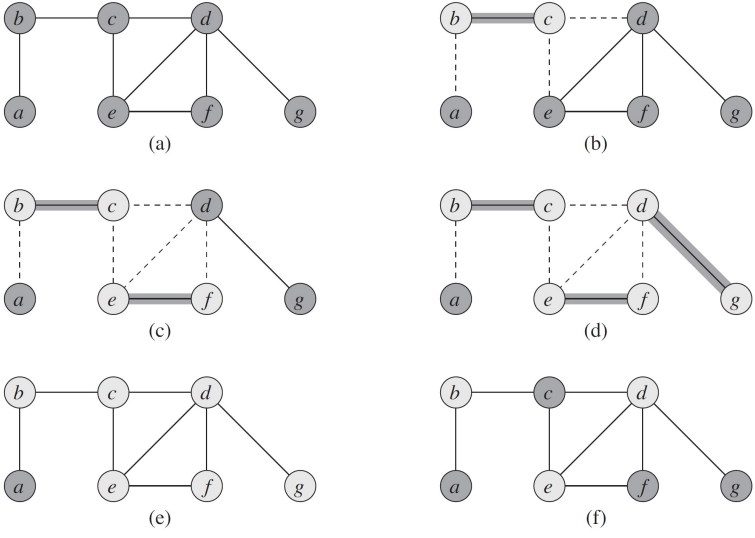
- **顶点覆盖的实质是用尽量少的顶点覆盖所有边**
- 作为对比，最小生成树是用权重和最小的边集连接所有顶点
- 顶点覆盖的规模是其中所包含的顶点数
- 顶点覆盖问题的目标是在一个给定的无向图中，找出一个具有最小规模的顶点覆盖，即最优顶点覆盖
- 顶点覆盖问题是 NP-完全问题，找到最优顶点覆盖很难，但可以较容易的找到近似最优顶点覆盖

顶点覆盖问题的近似解法

APPROX-VERTEX-COVER(G)

1: $C = \emptyset$
2: $E' = G.E$
3: **while** $E' \neq \emptyset$ **do**
4: 　　任取 $(u, v) \in E'$
5: 　　$C = C \cup \{u, v\}$
6: 　　从 E' 中删除所有与 u、v 相连的边
7: **return** C

顶点覆盖算法运行实例



近似解：b, c, d, e, f, g；最优解：b, d, e!

顶点覆盖算法的性能分析

定理

APPROX-VERTEX-COVER 是一个多项式时间的 2 近似算法

证 1) APPROX-VERTEX-COVER 算法运行时间为 $O(V+E)$

2) 算法会循环直到 $E(G)$ 中的每条边都被 C 中某个顶点覆盖为止，所以算法一定会返回一个顶点覆盖

3) 设 A 为第4行所选择的边集，而在第6行所有与该边相连的边都会被删除而不会进入 A，所以不可能有 A 中的两条边由最优覆盖 C^* 中同一顶点覆盖 (C^* 中每个顶点只能覆盖 A 至多一条边，否则 A 中的两条边会相交于同一顶点)；又由于任意边必须被顶点覆盖，所以 $|C^*| \geq |A|$

4) 第4行选择的每条边会为 C 增加两个顶点，所以顶点覆盖的上界为 $|C| \leq 2|A| \leq 2|C^*|$，得证

装箱问题

装箱问题 (bin-packing problem)

装箱问题的输入是体积分别为 $a_1, a_2, \cdots, a_n \in (0, 1]$ 的 n 个物品和无穷个容积为 1 的箱子。要求输出 n 个物品的一个装箱方案，并使得所用箱子个数最少

- 实例：5 个物品体积为 0.3, 0.5, 0.8, 0.2, 0.4，最优装箱方案 {0.2, 0.8}、{0.3, 0.5}、{0.4}，共用掉 3 个箱子
- 实际应用：将 n 张尺寸不同的票券印刷在最少的具有标准尺寸的纸张上

装箱问题的近似算法

FirstFit-BinPack()

1: $k \leftarrow 1,\ B_1 \leftarrow \varnothing$
2: **for** $i \leftarrow 1$ **to** n **do**
3: 从箱子 $B_1, B_2, \cdots, B_k$ 中选择第一个能容纳物品 a_i 的箱子 B_j，$B_j \leftarrow B_j \bigcup \{a_i\}$
4: 若 B_j 不存在，则令 $B_{k+1} \leftarrow \{a_i\}, k \leftarrow k+1$
5: **return** $B_1, B_2, \cdots, B_k$

算法复杂度为 $O(n^2)$，因为算法在处理物品 i 时，最坏情况下需要检查 $B_1, B_2, \cdots, B_k$ 中的每个箱子，而此时 $k \leq i-1$，因此 $T(n) \leq \sum_{1 \leq i \leq n} (i-1) = O(n^2)$

装箱问题近似算法的近似比

定理

FirstFit-BinPack 是一个多项式时间的 2 近似算法。

证 1) 设 k 和 k^* 分别表示近似解和优化解所用箱子个数

2) $k^* \geq \sum_{1 \leq i \leq n} a_i$，和式为 n 个物品最少需占据的箱子的数量

3) 假设近似解 $B_1, B_2, \cdots, B_k$ 的箱子内物品体积分别为 $|B_1|, |B_2|, \cdots, |B_k|$，则 $|B_1| + |B_2| + \cdots + |B_k| = \sum_{1 \leq i \leq n} a_i$

4) 注意 $|B_i| + |B_j| > 1$ 对任选的两个箱子成立，否则算法会将这两个箱子的物品合并为一箱

5) 因此

$$\begin{aligned} k/2 &< (|B_1| + |B_2|)/2 + (|B_2| + |B_3|)/2 + \cdots \\ &\quad + (|B_{k-1}| + |B_k|)/2 + (|B_k| + |B_1|)/2 \\ &= |B_1| + |B_2| + \cdots + |B_k| = \sum_{1 \leq i \leq n} a_i \leq k^* \end{aligned}$$

最短并行调度问题

最短并行调度问题

有 m 台完全一样的机器和运行时长分别为 $t_1, t_2, \cdots, t_n$ 的任务，每个任务均需要排他性地在一台机器上运行。问题是求 n 个任务在 m 台机器上的一个调度序列，使得这些任务的并行时间最短，即尽早结束所有任务

求解策略

任意排定所有计算任务的一个序列，然后顺序将每个任务分配到 m 台机器中的一台。假设前 $i-1$ 项任务分配后各台机器上任务的合计总时间分别为 $T_1, \cdots, T_m$，则第 i 项任务分配给总时间最短的机器 j，即 $t_j = \min_{1 \le k \le m} T_k$

最短并行调度算法

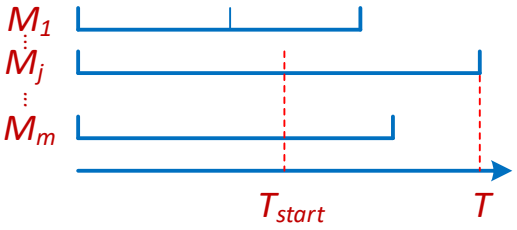
MinMakesspan-Scheduling($m, t_1, t_2, \cdots, t_n$)

1: 任意排定所有计算任务的一个顺序 $t_1, \cdots, t_n$
2: $T_k \leftarrow 0$; $M_k \leftarrow 0$; 其中 $k \in [1, m]$
3: **for** $i \leftarrow 1$ **to** n **do**
4: 找出 j 使得 $T_j = \min_{1 \le k \le m} T_k$
5: $T_j \leftarrow T_j + t_i$; $M_j \leftarrow M_j \bigcup \{i\}$
6: **return** $M_1, M_2, \cdots, M_m$

时间复杂度

第 4 行如能在常数时间内完成，则时间复杂度为 $O(n)$

最短并行调度算法的近似比



解 1) 用 T^* 和 T 分别表示优化解及近似算法的近似解的 (并行) 时间

2) $T^* \geq \left(\sum_{i=1}^{n} t_i\right)/m$，即所有任务同时结束所耗时间最短

3) $T^* \geq t_i$ 对任意 $i \in [1, n]$ 成立，因为每个任务都要在某台机器上处理

4) 设近似解中第 j 台机器处理时间最长且其处理的最后任务是 h，将任务 h 在第 j 台机器处理的开始时间为 T_{start}，易知 $T = T_{start} + t_h$，且 $T_s \leq \left(\sum_{i=1}^{n} t_i\right)/m \leq T^*$(否则所有机器的完成时间都大于平均时间 (注意 j 、h 的标准))，由此 $T = T_{start} + t_h \leq T^* + T^* = 2T^*$

旅行商问题简介

旅行商问题 (非形式化)

旅行商问题，又称为旅行推销员问题，指一个推销员旅行 n 个城市开展业务，城市之间的距离决定了其旅行的成本，要求给出该推销员的一个优化访问路线，使得每个城市仅被拜访一次，并使旅行成本最低

旅行商问题 (形式化)

输入一个完全无向图 $G = (V, E)$，其中每条边 $(u, v) \in E$ 都有一个非负的整数代价 $c(u, v)$，求 G 的一条具有最小代价的哈密顿回路

三角不等式

设 $c(s, t)$ 表示从 s 到 t 的代价，三角不等式要求对三个点 $u, v, w \in V$，有：$c(u, w) \leq c(u, v) + c(v, w)$

本节只研究符合三角不等式的旅行商问题，如果不符合三角不等式，则不存在具有固定近似比的多项式时间近似算法

最小生成树问题

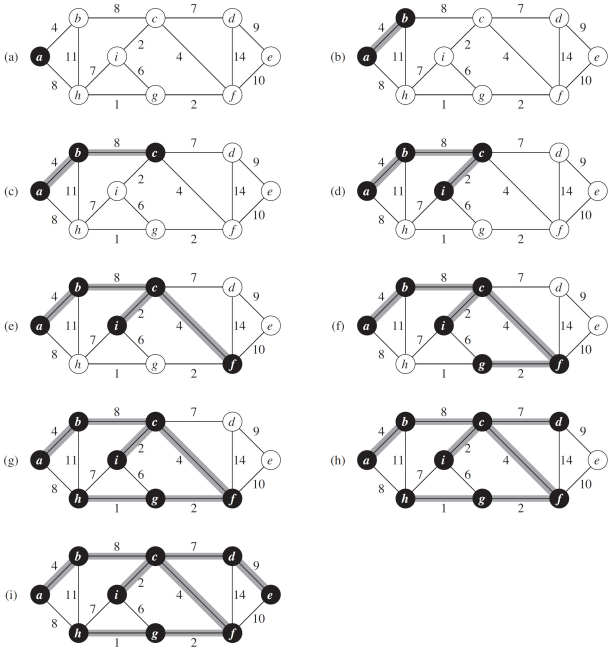
最小生成树

给定连通无向图 $G = (V, E)$，对 $\forall (u, v) \in E$，$w(u, v)$ 为其权重，寻找边集 $T \subseteq E$，使得 T 连通了所有的 V，且 $w(T) = \sum_{(u,v)\in T} w(u, v)$ 的值最小。T 必然组成一棵树，称求解具有该最小代价的问题为最小生成树问题

MST-PRIM(G, w, r)

```
1: for each u ∈ G.V do
2:     u : key = ∞
3:     u : π = NIL
4: r : key = 0
5: Q = G.V
6: while Q ≠ ∅ do
7:     u = EXTRACT-MIN(Q)
8:     for each v ∈ G.Adj[u] do
9:         if v ∈ Q and w(u, v) < v.key then
10:            v.π = u
11:            v.key = w(u, v)
```

MST-PRIM 算法运行实例



旅行商问题的近似解法

APPROX-TSP-TOUR(G, c)

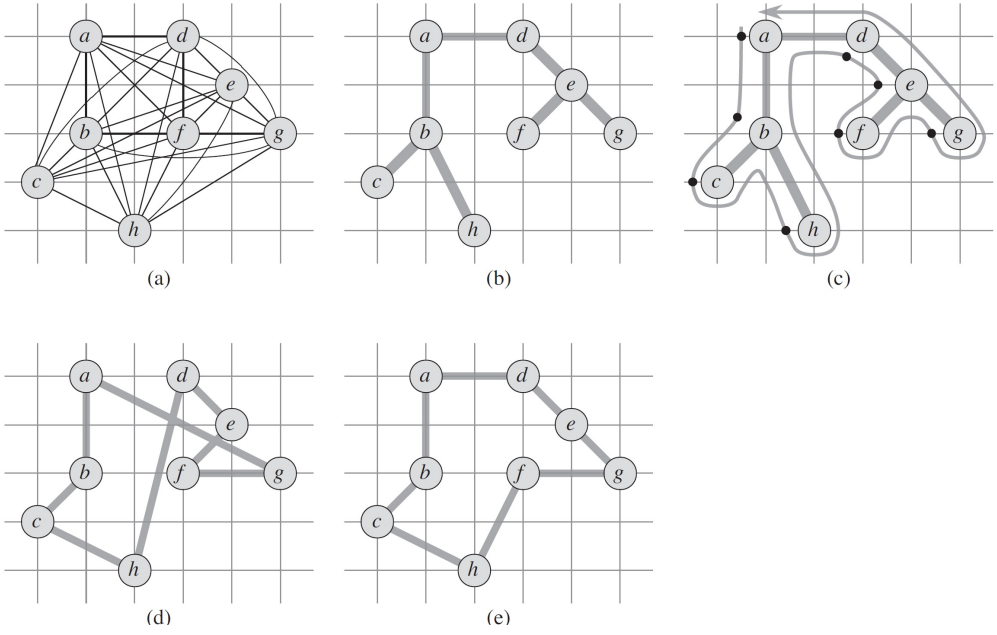
1: 任选 $r \in G.V$ 作为树的根
2: 调用 MST-PRIM(G, c, r) 产生最小生成树 T
3: 先序遍历 T 形成有序结点列表 L
4: 按照 L 的结点顺序访问各结点，形成哈密顿回路

性能分析：

- 第 2 行：$O(E + V \lg V) = O(V^2 + V \lg V) = O(V^2)$；
- 第 3 行：$O(E) = O(V^2)$，因为 G 是完全图；
- 第 4 行：$O(V)$；
- 所以总的时间复杂度为：$O(V^2)$。

在现代优化算法中将再次给出旅行商问题的更好解法！

旅行商问题求解实例



旅行商问题近似算法的近似比

定理

APPROX-TSP-TOUR 是一个用于解决满足三角不等式的旅行商问题的多项式时间 2 近似算法。

证 1) 设最优解哈密顿回路代价 H^*

2) 从 H^* 删除任何一条边可得到生成树，所以定有最小生成树 T 的权值满足：$c(T) \leq c(H^*)$

3) 对 T 的完全遍历 W 会经过 T 的每条边 2 次，所以 $c(W) = 2c(T) \leq 2c(H^*)$

4) 从 W 删除重复结点可最终得到一个解 H，由于满足三角不等式，所以删除结点会导致更短的路径，从而 $c(H) \leq c(W) \leq 2c(H^*)$，得证

专业相关扩展：EDA 顶会中的 TSP 问题

OPTIMIZATION RESULTS (BOLD FONT INDICATES BETTER RESULT. $K_1 = 1$ CORRESPONDS TO THE CONVENTIONAL METHOD.)

Problem	K_1	K_2	K_3	Best	Ave.	t_c [ms]	t_a [s]	t_t [s]	#Spins
burma14	1	-	-	31.83 (1.03)	37.01 (1.20)	-	1.179	1.184	196
burma14	4	-	-	**30.88** (1.00)	**32.51** (1.05)	1.321	0.255	0.262	63.82
ulysses16	1	-	-	75.03 (1.01)	88.30 (1.19)	-	1.798	1.807	256
ulysses16	4	-	-	**73.99** (1.00)	**79.76** (1.08)	1.306	0.455	0.464	97.98
ulysses22	1	-	-	89.90 (1.19)	108.66 (1.44)	-	5.019	5.035	484
ulysses22	6	-	-	**75.67** (1.00)	**78.94** (1.04)	1.534	0.723	0.741	117.22
bays29	1	-	-	13977.50 (1.50)	16181.28 (1.74)	-	12.623	12.652	841
bays29	6	-	-	**9155.45** (1.00)	**9878.29** (1.06)	1.628	1.203	1.222	154.44
berlin52	1	-	-	16852.80 (2.23)	19416.83 (2.57)	-	114.520	114.631	2704
berlin52	24	12	6	**8046.29** (1.07)	**9137.48** (1.21)	7.229	3.724	3.830	188.12
eil76	1	-	-	1532.65 (2.81)	1694.31 (3.11)	-	443.122	443.456	5776
eil76	24	12	6	**590.21** (1.08)	**642.21** (1.18)	7.305	8.257	8.366	285.16
eil101	1	-	-	2125.16 (3.33)	2391.28 (3.37)	-	1226.331	1227.032	10201
eil101	24	12	6	**717.54** (1.12)	**786.47** (1.22)	7.389	16.522	16.636	531.22

Ising 模型求解器因其找到组合优化问题近似解的效率而受到越来越多的关注，本文提出了一种递归聚类方法，可以加快 Ising 模型的计算速度并获得高质量的解

来源：DAC2020 论文“Clustering Approach for Solving Traveling Salesman Problems via Ising Model Based Solver”

子集和问题简介

子集和问题形式定义

- 输入：$< S, t >$，$S = \{x_1, x_2, \cdots, x_n\}$，$x_i$ 和 t 都是整数；
- 输出：S 的子集 A，满足：

$$\begin{aligned} A &\subseteq S \\ \sum_{x \in A} x &\leq t \\ \sum_{x \in A} x &= \max\{\sum_{x \in B} x | B \subseteq S\} \end{aligned}$$

子集和问题典型实例：装箱 (背包) 问题；将 0/1 背包问题的单位价格定为 1 即得到子集和问题！

典型应用：给定 CPU 有 W 个空闲周期，如何调度给定的多个任务 (每个占用 w_i 个周期) 使得 CPU 空闲周期最少？

子集和问题的指数时间精确算法

EXACT-SUBSET-SUM(S, t)

1: $n = |S|$
2: $L_0 =< 0 >$
3: **for** $i \leftarrow 1$ to n **do**
4: $\quad L_i =$ MERGE-LISTS($L_{i-1}, L_{i-1} + x_i$)
5: $\quad$ 从 L_i 中删除任何大于 t 的元素
6: **return** L_n 中最大的元素

考虑第 4 步，$|L_i| = 2|L_{i-1}| = \cdots = 2^i$，如果 t 很大，复杂度为 $O(\sum_{i=1}^{n} 2^i) = O(2^{n+1})$。

$S = \{1, 4, 5\}$

$$
\begin{aligned}
L_1 &= \{0, 1\} \\
L_2 &= \{0, 1, 4, 5\} \\
L_3 &= \{0, 1, 4, 5(5), 6, 9, 10\}
\end{aligned}
$$

子集和问题的完全多项式时间算法

APPROX-SUBSET-SUM(S, t)

1: $n = |S|$, $L_0 = \langle 0 \rangle$
2: **for** $i \leftarrow 1$ to n **do**
3: $\quad L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)$
4: $\quad L_i = \text{TRIM}(L_i, \varepsilon/2n)$ ▷ 修整列表，去除相近元素
5: $\quad$ 从 L_i 中删除任何大于 t 的元素
6: **return** L_n 中最大的元素 z^*

TRIM(L, δ)

1: SORT(L)
2: $m = L.length$
3: $L' = \langle y_1 \rangle$, $last = y_1$
4: **for** $i \leftarrow 2$ to m **do**
5: $\quad$ **if** $y_i > last \cdot (1 + \delta)$ **then**
6: $\quad\quad L' = L' + \langle y_i \rangle$
7: $\quad\quad last = y_i$
8: **return** L'

子集和问题的近似模式

- 算法需要输入近似参数 ε，从而返回的解在最优解的 $1/(1+\varepsilon)$ 倍内；
- 算法修整采用 $\delta = \varepsilon/2n$，使得用满足 $\frac{y}{1+\delta} \le z \le y$ 的 z 代表 y；
- 修整实例：$\langle 10, 11, 20, 21, 22\rangle \Rightarrow \langle 10, 20\rangle$；
- 实际运行实例：$S = \langle 104, 102, 201, 101\rangle$，目标 $t = 308$，$\varepsilon = 0.4$，算法返回 $z^* = 302$，处于最优解 307 的 $\varepsilon = 40\%$ 之内，约为 2%。

定理

APPROX-SUBSET-SUM 是子集和问题的一个完全多项式时间近似模式

证明思想：

1) 证明优化解 y^* 满足 $y^*/z^* \le 1+\varepsilon$
2) 证明算法的运行时间既是 $1/\varepsilon$ 的多项式，又是输入规模 n 的多项式

子集和问题近似比的证明 *

$\{x_1, x_2, \cdots, x_i\}$ 的子集和的全集 P_i，对其中任意至多为 t 的子集和 y，存在一个 $z \in L_i$，使得：

$$\frac{y}{(1+\varepsilon/2n)^i} \le z \le y \quad \text{(归纳法证明)}$$

上式对 $y^* \in P_n$ 也成立，从而存在 $z \in L_n$，使得：

$$\frac{y^*}{(1+\varepsilon/2n)^n} \le z \le y^* \Longrightarrow \frac{y^*}{z} \le (1+\frac{\varepsilon}{2n})^n$$

对于算法返回的 z^*，由于 $z \le z^* \le t$，则近似比

$$\rho(n) = \frac{y^*}{z^*} \le \frac{y^*}{z} \le (1+\frac{\varepsilon}{2n})^n$$

考虑到 $\lim_{n\to\infty}(1+\varepsilon/2n)^n = e^{\varepsilon/2}$ 且 $(1+\varepsilon/2n)^n$ 的导数为正 (↗)，从而

$$(1+\varepsilon/2n)^n \le e^{\varepsilon/2} \le 1+\varepsilon/2+(\varepsilon/2)^2 \le 1+\varepsilon$$

从而近似比为 $1+\varepsilon$。

子集和问题运行时间的证明 (续)*

运行时间与 L_i 的长度成正比。修整后 L_i 的连续元素 z 和 z' 满足 $z'/z > 1+\varepsilon/2n$，即间隙因子至少 $1+\varepsilon/2n$。另外每个列表都包含了 0，可能还有 1，以及另外至多 $\lfloor \log_{1+\varepsilon/2n} t \rfloor$ 个其它值，从而 L_i 的元素个数至多为

$$\begin{aligned}
\log_{1+\varepsilon/2n} t + 2 &= \frac{\ln t}{\ln(1+\varepsilon/2n)} + 2 \\
&\le \frac{2n(1+\varepsilon/2n)\ln t}{\varepsilon} + 2 \quad \left(\frac{x}{1+x} \le \ln(1+x) \le x\right) \\
&< \frac{3n \ln t}{\varepsilon} + 2 \quad (0 < \varepsilon < 1)
\end{aligned}$$

该式既是输入规模 n 的多项式，也是 $1/\varepsilon$ 的多项式。

子集和问题动态规划与随机算法的比较

- 动态规划是可以用来求子集和问题的，只要将每个元素的单价设为 1 即转化为 0/1 背包问题
- 0/1 背包问题动态规划算法的复杂度为 $O(nw)$，如果用于子集和问题复杂度为 $O(nt)$，当 t 比较大的时候，比如 $S = \langle 2^{63} + 1, 2^{63} + 3, 2^{63} + 7\rangle$，$t = 2^{64} + 5$，复杂度也很高
- 动态规划算法求解背包问题和子集和问题，只是一个伪多项式时间的算法
- 以本节的近似算法求解上述问题会快的多！

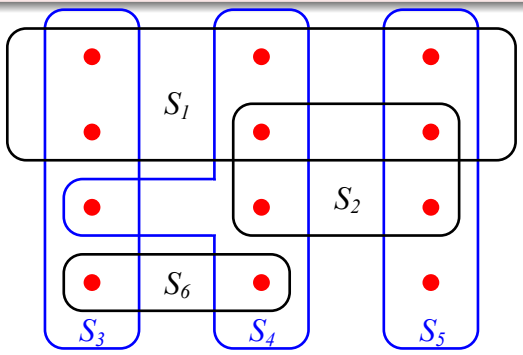
提纲

集合覆盖问题定义

集合覆盖问题

- 输入：有限集 X，X 的所有子集 F，$X = \cup_{S \in F} S$；
- 输出：$C \subseteq F$，满足 $X = \cup_{S \in C} S$ 且 $|C|$ 最小。

最小集合覆盖是很多实际问题的抽象，且是一个 NP 完全问题

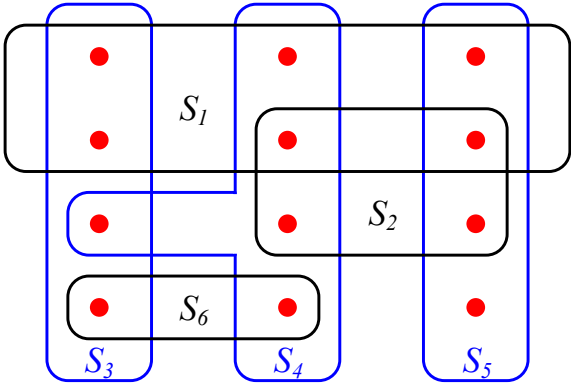


$F = \{S_1, S_2, S_3, S_4, S_5, S_6\}$，最优覆盖 $C = \{S_3, S_4, S_5\}$

集合覆盖近似算法的设计

算法思想

贪心策略：选择能覆盖尽量多未被覆盖元素的子集！



近似解 $C = \{S_1, S_4, S_5, S_3\}$

集合覆盖的贪心近似算法

GREEDY-SET-COVER(X, F)

1: $U = X, C = \emptyset$
2: **while** $U \neq \emptyset$ **do**
3: 选择一个 $S \in F$ 使得 $S \cap U$ 最大
4: $U = U - S$
5: $C = C \cup S$
6: **return** C

复杂度分析

- 此处的贪心算法无法保证得到最优解!
- 3 – 5 行的循环次数至多为 $\min(|X|, |F|)$;
- 计算 $S \cap U$ 需要时间 $O(|X|)$;
- 第 4 步需要时间 $O(|X||F|)$;
- 算法的时间复杂度为 $O(|X||F| \min(|X|, |F|))$。

集合覆盖算法的近似比 *

定理

GREEDY-SET-COVER 是一个多项式时间的 $\rho(n)$ 近似算法，$\rho(n) = H(\max\{|S| : S \in F\})$，$H(d)$ 是第 d 级调和级数 $H_d = \sum_{i=1}^{d} 1/i$，且 $H(0) = 0$

证 1) 已证明 GREEDY-SET-COVER 以多项式时间运行，只要再证明其具有 $\rho(n)$ 近似比即可。

2) 令 S_i 为选入的第 i 个子集，每个 S_i 并入 C 时赋予代价 1，并将这个代价 1 平分给首次被 S_i 覆盖的 X 的元素。对 $x \in X$，设 c_x 表示分配给元素 x 的代价。可知每个元素仅仅在其被首次覆盖时分配代价，从而如果 x 首次被 S_i 覆盖，我们有：

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$$

算法的每一步均会分配 1 单位的代价，因此：

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

集合覆盖算法的近似比 (续)*

由于最优覆盖 C^* 至少要包含 X 的所有点，从而：

$$\sum_{S\in C^*}\sum_{x\in S} c_x \geq \sum_{x\in X} c_x$$

综合前两式可得：

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

假设 $\sum_{x\in S} c_x \leq H(|S|)$ 成立，则可得：

$$|C| \leq \sum_{S\in C^*} H(|S|) \leq |C^*| \cdot H(\max\{|S| : S \in F\})$$

从而定理成立。关键是要证明蓝色部分的假设。

集合覆盖算法的近似比证明 (续)*

令 $u_i = |S - (S_1 \cup S_2 \cup \cdots \cup S_i)|$ 为前 i 个集合选出后，任意子集 $S \in F$ 中余下的未被前 i 个集合覆盖的元素的个数。定义 k 为使得 $u_k = 0$ 的最小下标，则可知 S 中会有 $u_{i-1} - u_i$ 个元素首次被 S_i 所覆盖。于是：

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} [(u_{i-1} - u_i) \cdot \frac{1}{|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}]$$

贪心策略保证了：

$$|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| \geq |S - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| = u_{i-1}$$

从而：

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

集合覆盖算法的近似比证明 (续)*

下面给出公式的界：

$$
\begin{aligned}
\sum_{x\in S} c_x &\le \sum_{i=1}^{k}(u_{i-1}-u_i)\cdot\frac{1}{u_{i-1}} \\
&= \sum_{i=1}^{k}\sum_{j=u_i+1}^{u_{i-1}}\frac{1}{u_{i-1}} \le \sum_{i=1}^{k}\sum_{j=u_i+1}^{u_{i-1}}\frac{1}{j} \qquad (\text{因为 } j \le u_{i-1}) \\
&= \sum_{i=1}^{k}\left(\sum_{j=1}^{u_{i-1}}\frac{1}{j}-\sum_{j=1}^{u_i}\frac{1}{j}\right) \\
&= \sum_{i=1}^{k}\big(H(u_{i-1})-H(u_i)\big) = H(u_0)-H(u_k) \\
&= H(u_0)-H(0) = H(u_0) = H(|S|)
\end{aligned}
$$

从而完成了对关键不等式的证明。

提纲

基于局部搜索的近似算法

基本思想

局部搜索广泛用于近似求解难解的优化问题，基本思路是从任意 (或特意选取的) 一个可行解出发，不断对其进行局部细微改动，使得优化问题的目标函数取值逐步达到最优，最后获得局部最优解作为近似解输出

局部搜索近似算法必须能够在多项式时间内终止，为此必须证明算法从任意可行解出发，至多经过多项式步操作即可获得一个局部最优解

最大割问题

最大割问题

最大割问题的输入是一个加权图 $G = (V, E)$，其中每条边 $(u, v) \in E$ 具有正整数权值 w_{uv}，求解顶点集的一个子集 S 使得 $\sum_{u \in S, v \in V - S} w_{uv}$ 达到最大值。

虽然最小割问题可以在多项式时间内求解 (最大流算法的对偶问题)，但最大割问题却是 NP-完全问题！

最大割问题的近似算法求解思路

- 首先初始化 $S=\{u\}$，u 是任选的一个顶点，此时介于 S 和 $V-S$ 间的所有边构成一个割，因此 S 称为问题的一个可行解
- 搜索思路：从 V 中选择一个顶点 v，交换 v 在 S 和 $V-S$ 中的位置，即如果 $v\in S$，则令 $S=S-\{v\}$，否则令 $S=S\bigcup\{v\}$
- 为了避免上面的局部搜索的盲目性，需要搜索能使得目标函数值增大，即要求交换能使得下述代价函数为正：

$$cost(\mathbf{v},S)=\begin{cases}\sum\limits_{u\in S}w_{u\mathbf{v}}-\sum\limits_{u\in V-S}w_{u\mathbf{v}} & \mathbf{v}\in S\\ \sum\limits_{u\in V-S}w_{u\mathbf{v}}-\sum\limits_{u\in S}w_{u\mathbf{v}} & \mathbf{v}\in V-S\end{cases}$$

代价函数成立的原因在于：将 $\mathbf{v}$ 换入 $V-S$，将新增 $S\to V-S$ 的割边，同时原有 $V-S\to S$ 的割边消失！

最大割近似算法

ApproxMaxCut($G=(V,E), W=\{w_{uv},(u,v)\in E\}$)

1: $S \leftarrow \{u\}$ ▷ u 是 V 中任意顶点
2: **repeat**
3: 任取一个满足 $cost(v,S)>0$ 的顶点 $v\in V$，交换 v 在 S 和 $V-S$ 中的位置
4: **until** 不存在 $v\in V$ 使得 $cost(v,S)>0$
5: **return** S

复杂度分析

第 3 步最坏情况下需要为任意 $v\in V$ 计算代价函数 $cost(v,S)$，而计算 $cost(v,S)$ 的最坏开销为 $|V|$，因此单次执行第 3 步的最坏开销为 $O(|V|^2)$(查找所有 V 才找到可交换点)；可能存在换过去又换回来的情况，但考虑第 3 步每执行一次代价至少增大 1(所有权重都是正整数)，而可行解的代价上界为 $\sum_{uv\in E} w_{uv}$，因此循环至多执行 $\sum_{uv\in E} w_{uv}$ 遍；最终算法的时间复杂度为 $O(|V|^2\sum_{uv\in E} w_{uv})$。

最大割近似算法的近似比为 2

证 用 S^* 和 S 分别表示问题的优化解和近似解。由于 S 是局部最优解，故 $\forall v \in S$ 必有 $cost(v, S) < 0$，即对某特定 $\mathbf{v}$：

$$\sum_{u\in S} w_{u\mathbf{v}} < \sum_{u\in V-S} w_{u\mathbf{v}} \Rightarrow \sum_{u\in V-S} w_{u\mathbf{v}} + \sum_{u\in S} w_{u\mathbf{v}} < 2\cdot\sum_{u\in V-S} w_{u\mathbf{v}}$$

$$\Rightarrow \frac{1}{2}\sum_{u\in V} w_{u\mathbf{v}} < \sum_{u\in V-S} w_{u\mathbf{v}} \quad (\forall \mathbf{v} \in S)$$

$$\text{同理：}\ \frac{1}{2}\sum_{u\in V} w_{u\mathbf{v}} < \sum_{u\in S} w_{u\mathbf{v}} \qquad (\forall \mathbf{v} \in V-S)$$

对上两式的特定 $\mathbf{v}$ 分别推广到所有 $v \in S$ 和 $v \in V-S$，则其右侧 $\sum_{u\in V-S, v\in S} w_{uv} = \sum_{u\in S, v\in V-S} w_{uv} = w(S)$。对推广后的两式左右分别相加，右侧为 $2\cdot w(S)$，左侧 (注意括号内 w_{uv} 重复加)：

$$1/2\cdot\Big(\sum_{u\in S, v\in S} + \sum_{u\in V-S, v\in S} + \sum_{u\in V-S, v\in V-S} + \sum_{u\in S, v\in V-S}\Big) = \sum_{uv\in E} w_{uv}$$

$$\Longrightarrow w(S^*) \le \sum_{uv\in E} w_{uv} < 2\cdot w(S)$$ (注意割集权重小于所有边权重之和)

小结

小结：

- 近似算法的思想及应用
- 近似算法的近似比分析方法 (了解)
- 能分析简单近似算法的近似比 (重点)