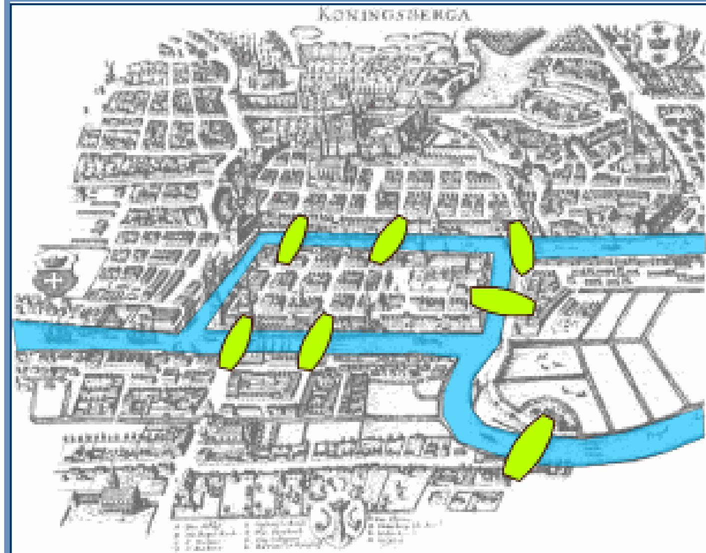


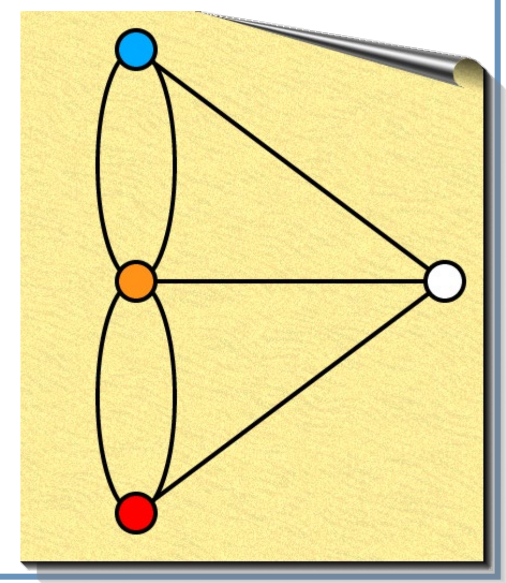
通信网络算法思维

Part 1: 困难问题的求解

——基于LP的近似算法

王晟

博士　教授　博导



基于LP的近似算法

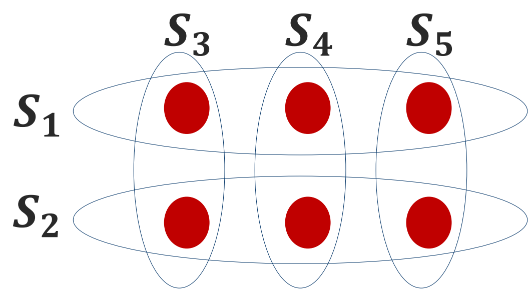
1. Set Cover
2. Min. Congestion : Rounding
3. MultiCut : Primal-Dual

SC: 基于LP的近似算法

1. Dual Fitting
2. Rounding
3. Primal-Dual

Set Cover(No Cost)

先来看一个简单的问题.



- 输入: m个子集$S_1, S_2, \dots, S_m \subseteq U$
- 目标: 挑选尽量少的子集来覆盖U.

Note:

- 与Set Coverage问题的区别: 约束和优化目标刚好反过来.
- 应用: 与Set Coverage的应用场合一样. 只是目标和约束不同.
- 困难的原因: 与Set Coverage也是一样的. 子集大小与冗余度的矛盾.

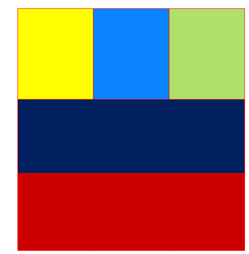
Set Coverage问题我们是怎么求解的?

➡ 很显然, 这里也可以用这个思路.

Greedy-SC-NoCost

算法:

- $\mathcal{C} = \emptyset$
- **While** $\mathcal{C}$不能覆盖U, **Do**
  挑选新覆盖元素最多的子集S_i, 加入$\mathcal{C}$.
- **return** $\mathcal{C}$.



坏例:

- $k = 3; |S_1| = |S_2| = |S_3| = \dfrac{|U|}{3};$
  $|S_4| = \left(\dfrac{1}{3} + \epsilon\right)|U|; |S_5| = \left(\dfrac{2}{9} + \epsilon\right)|U|$

$OPT = 3, GC = 5$

在Set Coverage中,这同时也是紧例. 但对Set Cover, 还有更坏的例子.

近似比

Claim-1: 令$n = |U|$, 则该算法ln$n$-近似.

- 首先注意到: 对同一实例, 运行过程与**Set Coverage**一样. 仅终止条件不同.

- **Set Coverage**问题中, 假如k个子集能覆盖的元素最多为x的话, 该算法保证循环k次后,能至少覆盖$\left(1-\frac{1}{e}\right)x$个元素.

- **Set Cover**问题中, 假如覆盖全部元素所需子集数目最少为y(即**OPT**), 即存在y个子集, 其覆盖的元素总和为$|U|$.

➡ 该算法前y次循环覆盖的元素数目至少为$\left(1-\frac{1}{e}\right)|U|$

? 再运行y次, 还剩多少元素未覆盖?

➡ 只剩$\frac{1}{e^2}|U|$

- $y\ln n$次循环后, 即可覆盖全集.

- 得证.

Set Cover (with cost)

略有区别:

- 输入: m个子集$S_1, S_2, \dots, S_m \subseteq U$, 子集代价为$c_i \geq 0, i = 1, 2, \dots, m$.
- 目标: 挑选总代价最小的多个子集来覆盖U.

➡ 换句话说,不仅希望选择的子集新覆盖元素多, 而且希望代价小.

➡ 贪心准则呼之欲出.

算法:

- $\mathcal{C} = \emptyset$
- While $\mathcal{C}$不能覆盖U, Do
  挑选r_i最小的子集S_i, 加入$\mathcal{C}$.
  $r_i = c_i/(\#新覆盖元素)$
- return $\mathcal{C}$.

Claim-2: 令$n = |U|$, 则该算法ln$n$-近似.

追踪进展

- 你不该奇怪: 我们的第一个念头是追踪进展.
- **Lemma-3:** 任给S_i, 假定当前贪心解已经覆盖了S_i中的l个元素, 则下一个被选中的子集的贪心指标最多为:

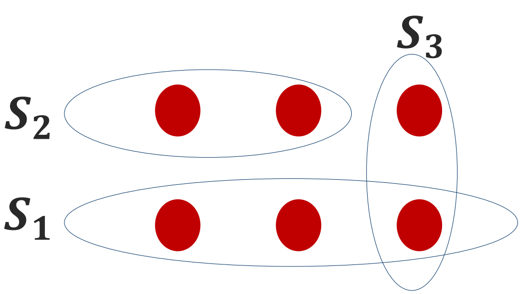
$$\frac{c_i}{|S_i| - l}$$

  - 这个式子描述的正是S_i的贪心指标.
  - 贪心算法选择最小贪心指标的子集加入解.
  - 得证.
- 为了利用该引理, 我们需要引入一个奇怪的变量.
  - 接下来, 我将着重揭示该变量“是什么”.
  - 关于“为什么”, 稍后会回来.

那个奇怪的变量…

- 针对每个元素$e \in U$, 定义q_e表示 “贪心算法运行过程中, 首个覆盖e的子集的贪心指标r_i”.

➡ 注意, 贪心算法最终会覆盖所有元素, 所以每个q_e都有良好定义.



$c_1 = c_2 = c_3 = 1$

- 第一次选中S_1, $r_1 = 1/3$,因此下排三个元素的q_e都是1/3.
- 第二次选中S_2, $r_2 = 1/2$,因此左上两个元素的q_e都是1/2.
- 第三次选中S_3, $r_3 = 1/1$,因此右上那个元素的q_e是1.

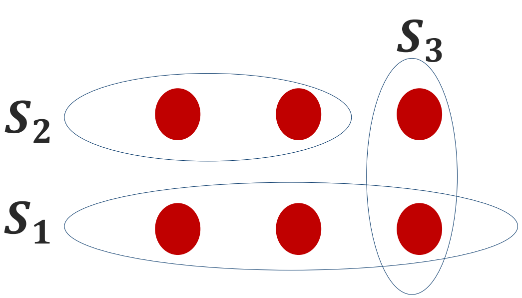
q值上限

Corollary-4: 任给S_i, 其中第j个被算法覆盖的元素e满足:

$$q_e \le \frac{c_i}{|S_i| - (j-1)}$$

Lemma-3: 任给S_i, 假定当前贪心解已经覆盖了S_i中的l个元素, 则下一个被选中的子集的贪心指标最多为:$\frac{c_i}{|S_i|-l}$

- 若S_i中的元素一个一个被覆盖, 那么该结论很明显.
- 对于“一批一批”被覆盖的情形, 该结论“更加满足”.



$c_1 = c_2 = c_3 = 1$

- 该结论说, S_1中的三个q_e上限分别为1/3, 1/2和1, 事实上, 它们的取值都是1/3.
- 该结论要求 S_3中的两个q_e上限分别为1/2和1, 事实上, 它们的取值是1/3和1.

子集上限

由q值上限[Corollary-4], 容易得到下面的“子集上限”：

$$\sum_{e\in S_i} q_e \le \frac{c_i}{|S_i|} + \frac{c_i}{|S_i|-1} + \cdots + \frac{c_i}{2} + c_i$$

$$\approx c_i \ln|S_i|$$

$$\le c_i \ln n, \qquad \text{其中} n = |U|$$

- 调和级数求和是通过积分近似的, 结果中还有一个小于1的常数(欧拉常数), 忽略该常数误差不大.
- 最后一步的放大其实并无必要. 定义$max_i|S_i|$为上界更精确, 只是不那么简洁.

全集代价

- 由q值定义, 也很容易得到下面的“全集代价”：

$$\sum_{e\in U} q_e = \text{贪心解的代价}$$

- 应用归纳法于贪心算法求解过程.
- 初始化时, 所有q_e都为0; 尚未挑选子集,故RHS也为0.
- 运行过程中, 假如挑选了子集S_i, 显然RHS会增加c_i.
- 由于新覆盖了一些元素, 其q值会变为非零, 故LHS也会增加.

LHS增加多少?

也是c_i, 因为: $r_i(\#\text{新覆盖元素}) = c_i$

证明Claim-2

Claim-2: 令$n = |U|$, 则该算法ln$n$-近似.

- 令$\{S_1^*, \dots, S_k^*\}$表示最佳Set Cover, 而OPT表示其代价.
- 由前面的讨论有:

$$
\begin{aligned}
\text{贪心解代价} &= \sum_{e \in U} q_e && \text{全集代价}\\
&\le \sum_{i=1}^{k} \sum_{e \in S_i^*} q_e && \text{同一元素可能存在于多个子集中}\\
&\le \sum_{i=1}^{k} c_i \ln n && \text{子集上限}\\
&= OPT \ln n && \text{OPT的定义}
\end{aligned}
$$

- 得证.

紧例

紧例:

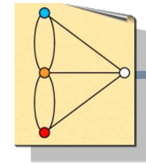
- $U = \{1, 2, \ldots, n\}$
- $S_0 = U, c_0 = 1 + \epsilon$
- $S_i = \{i\}, c_i = \dfrac{1}{i}, i = 1, 2, \ldots, n$

➡ 最佳解显然应是S_0,最佳值为$1 + \epsilon$.

➡ 贪心解的挑选顺序为$S_n, S_{n-1}, \ldots, S_1$, 总代价$\approx \ln n$.

我们的分析没有改进的空间.

那朵疑云...



至此, 我们应该有这样的印象:

- 这是个符合直觉的贪心算法;
- 其证明逻辑完美无缺.

只是, q值是怎么想到的?
面对一个新的算法,我能不能想到?

- 这个q值是对偶变量.
- 整个证明过程是在“Dual-Fitting”的思路下构造的.
- 我们接下来就用LP对偶理论来重新理解这个证明过程.

ILP vs. LPR

SC-ILP

$$\text{Minimize} \sum_{i=1}^{m} c_i x_i$$

subject to

$$\sum_{i:e\in S_i} x_i \geq 1, \qquad \forall e \in U$$

$$x_i \in \{0,1\}, \qquad \forall S_i$$

SC-LPR

$$\text{Minimize} \sum_{i=1}^{m} c_i x_i$$

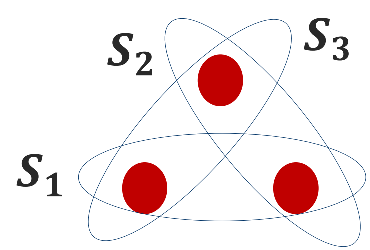
subject to

$$\sum_{i:e\in S_i} x_i \geq 1, \qquad \forall e \in U$$

$$x_i \geq 0, \qquad \forall S_i$$

回忆前面对该问题的LP建模.

- 松弛掉整数约束后, 分数解可能更好.



$c_1 = c_2 = c_3 = 1$

- 对该例, OPT=2, F-OPT=3/2

➡ FOPT $\leq$ OPT

LPR vs. LPR-D

- 回忆对偶推导的方法.
- 弱对偶定理怎么说的?
  - 对偶可行解的目标值是FOPT的下限.
- **Lemma-5: 令$\{p_e\}$是LPR-D的可行解, 则有:**

$$\sum_{e \in U} p_e \le \mathrm{FOPT} \le \mathrm{OPT}$$

SC-LPR

$$\text{Minimize} \quad \sum_{i=1}^{m} c_i x_i$$

subject to

$$\sum_{i: e \in S_i} x_i \ge 1, \qquad \forall e \in U$$

$$x_i \ge 0, \qquad \forall S_i$$

SC-LPR-D

$$\text{Maximize} \quad \sum_{e \in U} p_e$$

subject to

$$\sum_{e \in S_i} p_e \le c_i, \qquad \forall S_i$$

$$p_e \ge 0, \qquad \forall e \in U$$

Dual-Fitting

回忆前面的证明中, 我们用到的关于q值的两个结论:

- 子集上限: $\sum_{e\in S_i} q_e \le c_i \ln n$
  - ➡ 若定义p=q/ln$n$, 则p对偶可行.
- 全集代价: $\sum_{e\in U} q_e =$ 贪心解代价
  - ➡ 贪心解代价是p的目标值的ln$n$倍.

➡ 由Lemma-5, p的目标值是OPT下限.

SC-LPR-D

$$\text{Maximize} \sum_{e\in U} p_e$$

subject to

$$\sum_{e\in S_i} p_e \le c_i, \qquad \forall S_i$$

$$p_e \ge 0, \qquad \forall e \in U$$

重述这个逻辑:

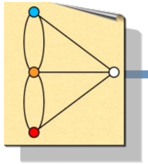
- A)根据对偶解的含义构造变量;[q]
  B)它可以刻画算法的目标值;[全集代价]
  C)讨论它与对偶可行条件的差距;[ln$n$]
  D)建立算法目标值与FOPT的差距; [ln$n$]

! 这就是Dual-Fitting证明思路.

SC: 基于LP的近似算法

1. Dual Fitting
2. Rounding
3. Primal-Dual

Vertex Cover(with cost)



- 与我们最早讨论的No Cost版本相比, 有两个区别:
  - 图中每个顶点都附有一个给定的代价, $c_v, \forall v \in V$.
  - 目标是用最小代价覆盖所有边.
- 显然, 该问题归约为Set Cover with Cost问题.
  - VC实例中的边对应SC实例中的元素;
  - VC实例中的顶点对应SC实例中的子集;
    - 特殊之处: 每个元素在这些子集中出现的“频率”都是2.
- 这个归约关系意味着什么?

意味着...

- 求解SC的算法可以用来求解VC问题.
  - 不需实例变换, 直接根据SC的贪心思想来设计VC算法?
- VC是SC的特例, 可以期待更好的算法.
  - 很遗憾, 贪心得到的算法近似比不是常数.
- 好消息: VC-with Cost 有常数近似比的算法.
  - 用LP-Rounding技术就可以得到.

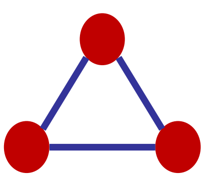
VC: LP-Rounding

算法:

- 调用LP-Solver求出LPR的最佳解$\mathbf{x}^*$.
- Return $S = \left\{v \in V : x_v^* \geq \frac{1}{2}\right\}$

回忆:

- LPR的解不保证整数; 但往往目标值更好.



$c_v = 1$

$OPT = 2$

$FOPT = 3/2$

Rounding$= 3$

VC-ILP

$$\text{Minimize} \sum_{v \in V} c_v x_v$$

subject to

$$x_v + x_w \geq 1, \qquad \forall e(v, w) \in E$$
$$x_v \in \{0, 1\}, \qquad \forall v \in V$$

VC-LPR

$$\text{Minimize} \sum_{v \in V} c_v x_v$$

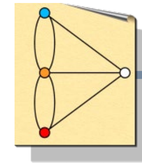
subject to

$$x_v + x_w \geq 1, \qquad \forall e(v, w) \in E$$
$$x_v \geq 0, \qquad \forall v \in V$$

LP-Rounding的特点

- 思路上很容易理解:
  - 求解LPR得到的是“比最佳解更好的解”,但它不可行.
    - 类比: 求解MTSP问题时构造的MST.
  - 把它变成可行的整数解就可以了.
    - 类比: 求解MTSP问题时, Tree-Doubling + Shortcut
- 细节上不容易处理:
  - Rounding的方式并不简单: 要保证可行.
    - VC是个Happy Case.

Rounding算法2-近似



Lemma-6: 上述LP-Rounding算法输出的S一定是可行解.

- $\mathbf{x}^*$是LPR的最佳解, 必满足约束: $x_v^* + x_w^* \geq 1, \forall(v,w) \in E$
  - 换句话说, 每条边的两个关联顶点中, 至少有一个在最佳解中取得超过1/2的解.
  - 算法保证: 每条边都至少有一个顶点被加入S.
- 得证.

Claim-7: 上述LP-Rounding算法是2-近似算法.

- $$\sum_{v\in S} c_v \leq \sum_{v\in V} c_v(2x_v^*)$$
  $$= 2\,FOPT$$
  $$\leq 2\,OPT$$
  - 第一个不等式: S中的顶点满足$2x_v^* \geq 1$, V–S中的顶点对RHS贡献非负.
  - 等式: $\mathbf{x}^*$是LPR的最佳解.
  - 第二个不等式: LPR vs. ILP
- 得证.

真的是神器吗?

想想我们已经知道的事实:

- NP-H优化问题大多可以建模为ILP;
- 求解对应的LPR是多项式时间的;
- 把分数解用最靠近的整数代替并不费事.

➡ LP-Rounding简直是近似算法设计的神器!

没有免费午餐:

- 简单的“取整”方式可能导致不可行;
- 或者近似比难以分析, 甚至近似比很差.

➡ 我们接下来用SC问题为例.

SC: Rounding算法#1

算法#1: 取整方式: 四舍五入.

- 调用LP-Solver求出SC-LPR的最佳解$\mathbf{x}^*$.
- Return $S = \left\{S_i : x_i^* \geq \frac{1}{2}\right\}$ 至少对VC, 该算法成功.

考虑下面的问题实例:

- $U = \{a, b, c, d, e, f, g, h\}$
- $S_1 = \{a, b, c, d\}, S_2 = \{e, f, g, h\}, S_3 = \{a, b, e, f\}, S_4 = \{c, d, g, h\}$
  $S_5 = \{a, c, e, g\}, S_6 = \{b, d, f, h\}, \quad c_i = 1, \forall i = 1, \ldots, 6$

注意元素的出现模式: 每个元素的“频率”都是3.

LPR的最佳解为: $x_i^* = \frac{1}{3}, \forall i = 1, \ldots, 6$, FOPT=2

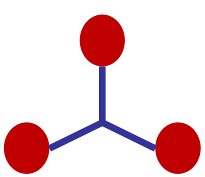
算法#1的运行结果? 全部取0, 不可行解.

SC: Rounding算法#2

- 算法#2: ➡ 取整方式: 零舍全入.
  - 调用LP-Solver求出SC-LPR的最佳解$\mathbf{x}^*$.
  - Return $S = \{S_i : x_i^* > 0\}$ ➡ 注意: 这样一定是可行解.
- 上一页中给的那个实例足以说明, 算法#2的近似比不可能小于$\Omega(n)$.

➡ 为了说明这一点, 需要理解该实例的构造方法: 超图.

- 超图(Hypergraph): 这种图由顶点和超边构成.
- 超边(Hyperedge): 一条超边可以与2个以上的顶点关联.



➡ 左图就是一个超图: 3个顶点, 1条超边.

算法#2的坏例

超图构造方式:

- 图中顶点共有$n = mk$个. 等分为不相交的k组,每组m个顶点.
  - ➡ 这k个不同的点集表示为: $V_1, V_2, \dots, V_k$
- 每条超边都与k个顶点关联. 选择方式: 从每个点集V_i中各选一个.
  - ➡ 该图中共计有m^k条超边.

与Set Cover实例的关系:

- 每条超边对应一个元素.
  - ➡ $|U| = m^k$
- 每个顶点对应一个子集.
  - ➡ 显然, 每个元素的频率都是k.
  - ➡ 前面那个实例: $m = 2, k = 3$

? 该实例的最佳解?

➡ 选择V_1对应的那m个子集. OPT=m

? LPR的最佳解?

➡ $x^* = 1/k$, FOPT=m

? 算法#2的解?

➡ 所有子集都选, mk

➡ 比最优解差$k = \Omega(n)$倍.

SC: Rounding算法#3

SC-LPR

$$\text{Minimize} \sum_{i=1}^{m} c_i x_i$$

subject to

$$\sum_{i:e\in S_i} x_i \geq 1, \qquad \forall e \in U$$

$$x_i \geq 0, \qquad \forall S_i$$

- 算法#3: ➡ 取整方式: 最大频率f.
  - 调用LP-Solver求出SC-LPR的最佳解$\mathbf{x}^*$.
  - Return $S = \{S_i: x_i^* \geq 1/f\}$

- **Lemma-8: Rounding算法#3输出的S一定是可行解.**
  - $\mathbf{x}^*$是LPR的最佳解, 必满足约束: $\sum_{i:e\in S_i} x_i^* \geq 1, \forall e \in U$

➡ 换句话说, 包含了元素e的多个子集(最多f个), 至少有一个子集在最佳解中取得超过1/f的解.

➡ 算法保证: 对每个元素,都至少有一个包含它的子集被加入S.

算法#3是f-近似的

Claim-9: Rounding算法#3是f-近似算法.

- $$\sum_{S_i \in S} c_i \le \sum_{i=1}^{m} c_i(f x_i^*)$$
  $$= f \cdot FOPT$$
  $$\le f \cdot OPT$$

  - 第一个不等式: 选入S中的子集必然满足$f x_v^* \ge 1$; 其他子集非负.
  - 等式: x^*是LPR的最佳解.
  - 第二个不等式: LPR vs. ILP

- 得证.

这个近似比有多好?

- 其实不怎么样. 算法#2的那个坏例, 刚好是这个算法的紧例. [那个例子中的k, 其实就是频率. 别忘了$k = \Omega(n)$]
- 接下来将证明, 算法#2其实也是f近似的.[当然证明没这么容易]
- 算法#3在现实中效果明显更好, 但在近似比的意义上并无区别.

算法#2也是

Claim-10: Rounding算法#2也是f-近似算法.

SC-LPR-D

$$\text{Maximize} \sum_{e\in U} p_e$$

subject to

$$\sum_{e\in S_i} p_e \le c_i, \qquad \forall S_i$$

$$p_e \ge 0, \qquad \forall e \in U$$

- 令x^*, p^*表示LPR和LPR-D的最佳解.

➡ 互补松弛条件:$\forall x_i^* \neq 0, \sum_{e\in S_i} p_e^* = c_i$

- 算法#2将所有非零x_i^*对应的子集S_i都放入了最终解S中.

➡ $$\sum_{S_i\in S} c_i = \sum_{S_i\in S}\sum_{e\in S_i} p_e^* \le \sum_{i=1}^{m}\sum_{e\in S_i} p_e^* \le f\sum_{e\in U} p_e^* = f\cdot FOPT \le f\cdot OPT$$

互补松弛条件　其他子集非负　e最多出现f次　强对偶定理　LPR是下界

- 得证.

小结LP-Rounding

- 确实是设计近似算法的神器.
- 但取整方案的选择并不容易.
  - 该选择会影响可行性. [算法#1]
  - 也可能会影响性能分析的难度. [算法#2]
  - 有时还可能会影响近似比. [算法#2不算]

Set Cover的故事还没完

- 对Set Cover问题, LP-Rounding提供的近似比为f.
  - 取整方案的选择并不容易.
  - 最坏情况下,还不如贪心算法[ln$n$]
- 接下来我们用Randomized Rounding(随机取整)设计算法.
  - 取整方案简单明了.
  - 得到的近似比为ln$n$.
  - 但是(总有个但是): 该近似比是概率意义上的.

随机取整

基本思路:

- 对0-1规划来说，将LPR的最佳解看做概率是很自然的.
- 然后，依概率来决定某个变量是否为1.
  - 例如，若Set Cover的LPR最佳解中$x_i^* = 0.6$，则产生一个[0,1]内均匀分布的随机数b.
  - 若$b \in [0, 0.6]$，则子集S_i被选中，否则该子集不在最后的解中.

两个明显的问题:

- 能保证可行吗?
- 近似比还是“最坏性能界”的含义吗?

随机性的影响

- 能保证可行吗?
  - 不能. 这是随机算法的关键特点. 你只能去习惯这个概念.
  - 通过多次运行(独立取随机数)同一过程, 然后合并这些解, 可以减小不可行的概率.
  - 是的, 最后的结论也只是不可行的概率小到一定程度.
- 近似比还是"最坏性能界"的含义吗?
  - 不是. 是概率意义上的. [具体语义将结合例子阐述]
  - 但是, 这种性能界并不依赖于实例的分布, 只依赖于算法中产生随机数的分布.
  - 换句话说, 这种性能界仍然是对任意实例都成立的.

随机算法虽然难以理解, 但却是个真正的"Big Idea".

单次运行RR的结果(1/2)

先看RR算法的解的代价.

- 令x^*表示LPR的最佳解.
- 将其理解为概率, 并依据这些概率来决定是否选择对应的子集.
- 最终选中的那些子集构成集合$\mathcal{S}$. 其代价为$c(\mathcal{S})$.
- 则解代价的期望为:

$$E[c(\mathcal{S})]=\sum_{i=1}^{m} Pr[S_i\text{被选中}]\cdot c_i=\sum_{i=1}^{m} x_i^* c_i = FOPT$$

单次运行RR的结果(2/2)

接下来计算: 元素$a \in U$被$\mathcal{S}$覆盖的概率.

- 假定元素a包含在k个子集中. 这些子集被选中的概率为$x_1^*, x_2^*, \ldots, x_k^*$.
- 由于是最佳解, 所以必然满足: $x_1^* + \cdots + x_k^* \geq 1$
- 由基础微积分的知识, 我们知道: 在这个条件下, 使得$Pr[a$被$\mathcal{S}$覆盖$]$最小的取值方式是: $\forall i = 1, 2, \ldots, k, x_i^* = \frac{1}{k}$

➡ 故有:

$$Pr[a\text{被}\mathcal{S}\text{覆盖}] \geq 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e}$$

➡ 其中e是自然对数的底.

➡ 显然, a不被覆盖的概率为1/e.

多次运行RR的效果

- **IDEA: 多次独立运行RR, 即可增大可行概率.**
  - 假定独立运行m次, 每次得到的$\mathcal{S}$都合并起来,合集记为$\mathcal{S}'$.
    - ➡ 显然, $Pr[a$未被$\mathcal{S}'$覆盖$]$会减小为$1/e^m$.
    - ➡ $Pr[\mathcal{S}'$不可行$]$也会减小.
- **但是: 解的代价也会相应增大.**
  - 由于每个子集被选中的概率增大了m倍.
    - ➡ 显然, $E[c(\mathcal{S}')] = m \cdot E[c(\mathcal{S})] = m \cdot FOPT$
- 运行次数m该如何选择?

WHP

运行次数m该如何选择?

➡ 要保证: “with high probability, 元素a被$\mathcal{S}'$覆盖.”

WHP的概念:

- 我们说“WHP, 事件E发生”, 是指算法可以通过设置某个参数来保证$\forall\alpha \geq 1$, E发生的概率至少为$1 - O(\frac{1}{n^{\alpha}})$.
  [或者等价地, 不发生的概率最多为$O(\frac{1}{n^{\alpha}})$]

➡ 这意味着, 该算法在多项式时间内, 即可“Almost certainly”保证E的发生.

对我们的问题来说:

- 事件E=元素a被$\mathcal{S}'$覆盖
- 算法可设置参数=运行次数m

参数m的设置方案

参数m的设置:

- 为追求可行, 只需选择保证右式成立的m即可: $\left(\frac{1}{e}\right)^m \leq \mathrm{O}\left(\frac{1}{\mathrm{n}^\alpha}\right) \approx \frac{\beta}{n^\alpha}$

- 另一方面, 我们又希望m不要太大.[影响RT, 也影响近似比]

➡ 令$m = c\ln n$, 其中c是保证右式满足的最小常数: $\left(\frac{1}{e}\right)^{c\ln n} \leq \frac{\beta}{n^\alpha}$

这就是保证"WHP, $\mathcal{S}'$可行, 且代价最小"的参数选择方案

算法性能

可行性:

- 此时有: $Pr[a\text{未被}\mathcal{S}'\text{覆盖}] \le \left(\frac{1}{e}\right)^{c\ln n} \le \frac{\beta}{n^{\alpha}}$
- 对所有$a \in U$求和, 有: $Pr[\mathcal{S}'\text{不可行}] \le \frac{\beta}{n^{\alpha-1}}$

近似比:

- 显然有: $E[c(\mathcal{S}')] \le c\ln n \cdot FOPT$
- 应用Markov不等式$Pr[X \ge t] \le E[X]/t$, 其中$t = \frac{c}{\beta}\ln n \cdot FOPT$,
  得到: $Pr\left[c(\mathcal{S}') \ge \frac{c}{\beta}\ln n \cdot FOPT\right] \le \beta$

举例

- 假如调用者设定: $\alpha = 1, \beta = 1/4$

  - 则算法应设定c为满足右式的最小常数: $\left(\frac{1}{e}\right)^{c\ln n} \le \frac{1}{4n}$

  - 算法运行$m = c\ln n$次RR后得到的解$\mathcal{S}'$, 可以保证针对任意实例, 有:$Pr[\mathcal{S}'\text{不可行}] \le \frac{1}{4}$, 并且$Pr[c(\mathcal{S}') \ge 4c \cdot \ln n \cdot FOPT] \le \frac{1}{4}$

  ➡ 换句话说, $Pr[\mathcal{S}'\text{可行且代价} \le 4c \cdot \ln n \cdot FOPT] \ge \frac{1}{2}$

  - 注意到, 得到$\mathcal{S}'$后检查一遍上述两个条件是否满足,只需多项式时间.

  ➡ 只要不满足, 可以重新运行一遍整个算法, 得到另一个解$\mathcal{S}''$.

  ➡ 上述结论保证: 为得到可行且满足近似比的解, 平均只需2遍运行.

- 这就是“该RR算法近似比为$\ln n$”的全部含义.

小结: Rounding

- **Happy Case: LP-Rounding for VC**
  - 简单取整方案, **2**-近似.
- **Not so happy: LP-Rounding for SC**
  - 合适的取整方案, **f**-近似, 最坏情况$\boldsymbol{f} = \boldsymbol{\Omega}(\boldsymbol{n})$.
- **Happy but Weird: Randomized-Rounding for SC**
  - **WHP**可行, **O(ln*n*)**-近似. 但是概率意义上的结论.

取整是设计近似算法的利器, 但应用起来并不容易.

SC: 基于LP的近似算法

1. Dual Fitting
2. Rounding
3. Primal-Dual

VC: 减少运行时间

回忆: 基于Rounding的2-近似算法.

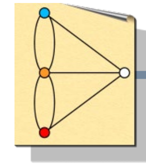
- 有人基于“unique games”猜想(一个强化版本的$P \neq NP$猜想), 证明了VC不可能有更好的近似比了.

➡ 看来, 改进近似比是不可能了.

- 但仍有改进的空间: 直接求解LP导致算法的运行时间不能令人满意.

➡ 接下来, 我们用Primal-Dual技术来设计一个高效算法.

VC: P-D schema



计划: 基于P-D schema来设计算法.

- 算法中始终保持对偶可行解, 追求主可行.
- 主可行的含义: 始终保持主解为整数,
  但直至算法结束, 才能覆盖所有元素.
- 两个互补松弛条件中, 一个被保持,
  另一个 “近似满足” .

➡ 要点是: 该算法不需显式求解LP,
但需要LP对偶理论的指导.

VC-ILP

$$\text{Minimize} \sum_{v\in V} c_v x_v$$

subject to

$$\begin{aligned} x_v + x_w &\geq 1, && \forall e(u,w)\in E \\ x_v &\in \{0,1\}, && \forall v\in V \end{aligned}$$

VC-LPR

$$\text{Minimize} \sum_{v\in V} c_v x_v$$

subject to

$$\begin{aligned} x_v + x_w &\geq 1, && \forall e(v,w)\in E \\ x_v &\geq 0, && \forall v\in V \end{aligned}$$

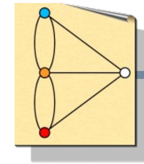
VC-LPR-D

$$\text{Maximize} \sum_{e\in E} p_e$$

subject to

$$\begin{aligned} \sum_{e\in\delta(v)} p_e &\leq c_v, && \forall v\in V \\ p_e &\geq 0, && \forall e\in E \end{aligned}$$

P-D算法 for VC



VC-P-D

- $\forall e \in E, p_e = 0$
- $\mathcal{S} = \emptyset$
- While $\mathcal{S}$未覆盖所有边 Do
  - 任选一条边$e = (v, w), v, w \notin \mathcal{S}$.
  - 增大p_e, 直至有对偶约束变紧.
  - 将紧约束对应的顶点加入$\mathcal{S}$.
- return $\mathcal{S}$

p_e涉及两个对偶约束. 分别对应顶点v, w.

“变紧”是指对偶约束取得等号.

VC-LPR-D

$$\text{Maximize} \sum_{e \in E} p_e$$

subject to

$$\sum_{e \in \delta(v)} p_e \le c_v, \qquad \forall v \in V$$

$$p_e \ge 0, \qquad \forall e \in E$$

如果不知道LP对偶, 这个算法完全不可能理解.

该算法提供的保障(1/2)

- $\forall e \in E, p_e = 0$
- $\mathcal{S} = \emptyset$
- **While** $\mathcal{S}$未覆盖所有边 **Do**
  - 任选一条边$e = (v, w), v, w \notin \mathcal{S}$.
  - 增大p_e, 直至有对偶约束变紧.
  - 将紧约束对应的顶点加入$\mathcal{S}$.
- **return** $\mathcal{S}$

该算法保证了以下几点:

- **(A)** 算法终止时, 得到的解$\mathcal{S}$可行.
- **(B)** p始终是LPR-D的可行解.
  - ➡ 初始化为0, 以后只会增大.
  - ➡ 每次增大都不违背对偶约束.
- **(C)** 若$v \in \mathcal{S}$, 则$\sum_{e \in \delta(v)} p_e = c_v$.
  - ➡ 每次加入$\mathcal{S}$都满足.
  - ➡ 这是主变量的互补松弛条件.

VC-LPR-D

$$\text{Maximize} \sum_{e \in E} p_e$$

subject to

$$\sum_{e \in \delta(v)} p_e \le c_v, \qquad \forall v \in V$$

$$p_e \ge 0, \qquad \forall e \in E$$

该算法提供的保障(2/2)

该算法保证了以下几点:

- (A) 算法终止时, 得到的解$\mathcal{S}$可行.
- (B) p始终是LPR-D的可行解.
- (C) 若$v \in \mathcal{S}$, 则$\sum_{e\in\delta(v)} p_e = c_v$.
- (D) 若$p_e \neq 0, e = (v, w)$,
  则$|\mathcal{S} \cap \{v, w\}| \leq 2$.

- $\forall e \in E, p_e = 0$
- $\mathcal{S} = \emptyset$
- While $\mathcal{S}$未覆盖所有边 Do
  - 任选一条边$e = (v, w), v, w \notin \mathcal{S}$.
  - 增大p_e, 直至有对偶约束变紧.
  - 将紧约束对应的顶点加入$\mathcal{S}$.
- return $\mathcal{S}$

➡ 最多两个端点在$\mathcal{S}$中, 即使$p_e = 0$,该条件也满足.

➡ 这实际上是对偶变量的互补松弛条件($|\mathcal{S} \cap \{v, w\}| = 1$)的 “近似版本” .

➡ 对偶变量的互补松弛条件近似满足, 不满足的程度不超过2倍.

VC-LPR

$$\text{Minimize} \sum_{v\in V} c_v x_v$$

subject to

$$x_v + x_w \geq 1, \quad \forall e(v,w) \in E$$
$$x_v \geq 0, \quad \forall v \in V$$

P-D设计原则

解读这些保证:

- (A) 算法终止时, 得到的解$\mathcal{S}$可行. ➡ 主可行.
- (B) p始终是LPR-D的可行解. ➡ 对偶可行.
- (C) 若$v \in \mathcal{S}$, 则$\sum_{e \in \delta(v)} p_e = c_v$. ➡ 满足第一个互补松弛条件
- (D) 若$p_e \neq 0, e = (v, w)$, 则$|\mathcal{S} \cap \{v, w\}| \leq 2$. ➡ 近似满足第二个

由LP对偶理论我们知道:

- 主可行 + 对偶可行 + 满足所有互补松弛条件 = 最佳解

➡ 对于NP-H问题, 我们显然不能指望这些条件同时满足.

不难得出针对NP问题的P-D算法设计原则:

- 保持对偶可行, 追求主可行.
- 保持互补松弛条件(一个或两个)近似满足.

➡ 近似程度就是最终算法的近似比.

近似比为2

Claim-11: 上述针对VC的P-D算法是2-近似算法.

- $$\sum_{v\in\mathcal{S}} c_v = \sum_{v\in\mathcal{S}} \sum_{e\in\delta(v)} p_e$$ 算法保证(C).

  $$= \sum_{e=(v,w)\in E} p_e \cdot |\mathcal{S} \cap \{v, w\}|$$ 交换求和顺序.

  $$\leq 2 \sum_{e\in E} p_e$$ 算法保证(D).

  $$\leq 2 \cdot FOPT$$ 算法保证(B); 弱对偶定理

  $$\leq 2 \cdot OPT$$ FOPT是下限

- 得证.

SC: P-D算法设计

应用上述原则，该算法应具备以下特点：

- 维护对偶变量p，始终保持其可行性.
- 维护主变量x，始终保持其为整数.
  并不断改进其可行性.直至最后覆盖所有元素.
- 算法中可以控制p的大小，并据此决定哪些x取得非零值.
  - ➡ 因此，主变量的互补松弛条件可以严格保证：$\forall x_i \neq 0, \sum_{e \in S_i} p_e = c_i$
- x的取值受控于p，不能保证对偶变量的互补松弛条件：$\forall p_e \neq 0, \sum_{i: e \in S_i} x_i = 1$
  - ➡ 但“不满足的程度”可以定界.
    即：$\forall p_e \neq 0, \sum_{i: e \in S_i} x_i \leq f$
  - ➡ 这将是整个算法的性能界.

SC-ILP

$$\text{Minimize} \sum_{i=1}^{m} c_i x_i$$

subject to

$$\sum_{i: e \in S_i} x_i \geq 1, \qquad \forall e \in U$$

$$x_i \in \{0, 1\}, \qquad \forall S_i$$

SC-LPR

$$\text{Minimize} \sum_{i=1}^{m} c_i x_i$$

subject to

$$\sum_{i: e \in S_i} x_i \geq 1, \qquad \forall e \in U$$

$$x_i \geq 0, \qquad \forall S_i$$

SC-LPR-D

$$\text{Maximize} \sum_{e \in U} p_e$$

subject to

$$\sum_{e \in S_i} p_e \leq c_i, \qquad \forall S_i$$

$$p_e \geq 0, \qquad \forall e \in U$$

P-D算法 for SC

SC-P-D

- $\forall e \in U, p_e = 0$
- $\mathcal{S} = \emptyset$
- While $\mathcal{S}$未覆盖所有元素 Do
  - 任选一个未覆盖元素e.
  - 增大p_e, 直至有对偶约束变紧.
  - 将紧约束对应的子集加入$\mathcal{S}$.
- return $\mathcal{S}$

SC-LPR-D

$$\text{Maximize} \sum_{e \in U} p_e$$

subject to

$$\sum_{e \in S_i} p_e \le c_i, \qquad \forall S_i$$

$$p_e \ge 0, \qquad \forall e \in U$$

显然，近似比为f

Claim-12: 上述针对SC的P-D算法是f-近似算法.

- $$\sum_{S_i \in \mathcal{S}} c_i = \sum_{S_i \in \mathcal{S}} \sum_{e \in S_i} p_e$$ 主变量互补松弛条件

  $$\leq \sum_{e \in U} f \cdot p_e$$ 对偶变量互补松弛条件

  $$\leq f \cdot FOPT$$ 对偶可行; 弱对偶定理

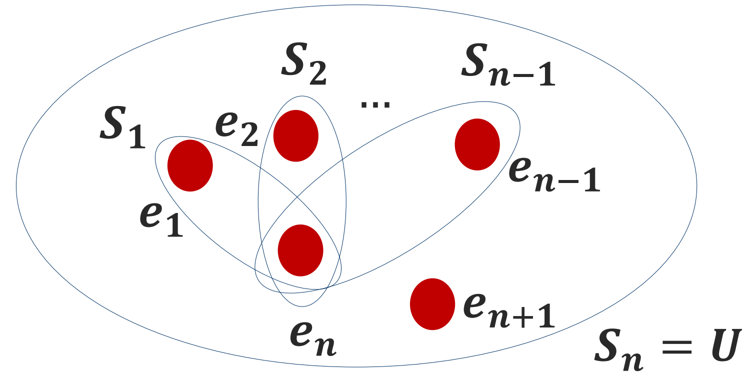
  $$\leq f \cdot OPT$$ FOPT是下限

- 得证.

紧例

考虑这样一族实例.

- 共计n+1个元素, n个子集.
- 子集定义和代价如右图所示.



$$c_1 = \cdots = c_{n-1} = 1$$
$$c_n = 1 + \epsilon$$

最大频率f=?

➡ 显然为n

- 假定算法第一次选中e_n, 当p_{e_n}增大到1时,$S_1, \dots, S_{n-1}$全部变紧, 加入$\mathcal{S}$.
- 下一步, $p_{e_{n+1}}$增大到ϵ时, S_n变紧, 加入$\mathcal{S}$.

➡ 算法终止时, 解中包含了全部子集, 代价为$n + \epsilon$.

最佳解呢?

➡ 只需S_n, 代价为$1 + \epsilon$.

Journey ahead

- 我们全面展示了基于LP的近似算法分析和设计技术.
  - Dual-Fitting: 分析近似比
    - 算法的设计与LP无关. 但有了LP对偶理论的帮助, 可以更系统地设计证明近似比的策略.
  - Rounding: 设计算法
    - 算法中直接求解LPR. 然后选择合适的取整方案, 确保可行性, 并据此分析近似比.
  - Primal-Dual Schema: 设计算法
    - 用LP对偶来指导算法的决策过程, 不用求解LP.
- 但仅用一个问题作为案例, 显然不够充分.
- 接下来: 讨论更多的问题及其近似算法.

基于LP的近似算法

1. Set Cover
2. Min. Congestion : Rounding
3. MultiCut : Primal-Dual

Rounding

1. 5 Tools for prob. analysis
2. Min-Congestion Routing

概率分析的工具

- 这部分介绍5个概率分析方面常用的结论.
  - 其中4个非常简单. 大多数几句话就可以证明.
  - 最后一个证明起来不容易.

  ➡ 所以我们重点展示其应用, 都不做证明.

- 如果你时常看见别人使用这些工具却难以明白为什么, 这部分内容大概正是你需要的.
- 本节内容讨论的例子跟近似算法无关.
- 最后一节再回到近似算法.

  ➡ 应用本节介绍的工具来分析一个随机取整算法.

工具#1

工具#1: 线性期望(Linearity of Expectation)

- 令$X_1, \dots, X_n$是定义在同一状态空间上的随机变量, 则有:

$$E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

注意:

- 证明很简单. 用期望的定义展开, 交换求和顺序.
- 不要求这些变量相互独立. 变量相乘就没有这么好的性质了.
- 算法分析中常用来分析复杂变量[例如Qsort中比较的次数]的期望. 因为这些复杂变量通常可以表达为一系列简单变量[指示变量]的求和.

工具#2

工具#2: Markov不等式

- 令X是非负随机变量, 均值有界. 则$\forall t \geq 1$, 有:

$$Pr\left[X \geq t \cdot E[X]\right] \leq 1/t$$

注意:

- 这是讨论“聚集程度”的工具. 表述的是“尾部概率”、偏移概率.
- 如果你只需要常数保证, 这个不等式通常就够了.
- 如果希望得到较好的结果, E[X]不能太大.
- 关键是, 该不等式对变量X的要求很少, 不需太多假设.

工具#3

工具#3: Chebyshev不等式

- 令X是非负随机变量, 均值有界, 且方差有界: $\sigma^2 = Var[X] < \infty$. 则$\forall t \geq 1$, 有:

$$Pr[|X - E[X]| > t \cdot \sigma] \leq 1/t^2$$

注意:

- 直接用**Markov**不等式即可证明.
- 如果除了均值, 你还能有效评估方差, 那么**Chebyshev**会得到比**Markov**更好的界.
- 但是别忘了, 对随机变量的要求更高了.

➡ 我们将来会看到, 当X为多个变量之和时, 这些变量两两独立才能有效评估方差.

工具#4

工具#4: Chernoff界

- 令$X_1, \ldots, X_n$是同一状态空间上的独立随机变量, 且都在[0,1]内取值, 令$X = \sum_{j=1}^{n} X_j$, 则:
  (a) $\forall \delta > 0$,
  $$Pr[X > (1+\delta) \cdot E[X]] < \left(\frac{e}{1+\delta}\right)^{(1+\delta)E[X]}$$
  (b) $\forall\ 0 < \delta < 1$,
  $$Pr\big[X < (1-\delta) \cdot E[X]\big] < e^{-\frac{\delta^2}{2}E[X]}$$

注意:

- 独立假设至关重要. 对随机变量取值也有要求[但适用于算法分析].
- 我们只用结论a, 但b也有很多应用.
- “指数上有东西”是这个工具强大的原因. [应用范围不止算法]
- 这是唯一一个证明起来不太容易的结论.

工具#5

工具#5: Union界(Boole不等式)

- 令$E_1, \ldots, E_k$表示k个事件, 则有

$$Pr[\text{至少其中之一发生}] \le \sum_{i=1}^{k} Pr[E_i]$$

注意:

- 这不是关于尾部概率的结论, 但通常与前三个工具一起使用.
- 证明起来同样很简单.
- 算法分析中, 通常关心的是坏事件. 只要每个坏事件概率不大, 而且坏事件的数目不多, 则我们可以说: 它们都不会发生.

应用案例#1:抛硬币

考虑以下案例.

- 抛n次硬币(独立实验).
- 目标: 我们希望求得参数t, 使得下式满足:

$$Pr[\text{至少有}t\text{次出现正面}] \le \frac{1}{n}$$

随机变量定义:

- 定义指示变量: $X_i = \begin{cases} 1, & \text{第}i\text{次出现正面} \\ 0, & \text{第}i\text{次出现背面} \end{cases}$
- 令 $X = \sum_{i=1}^{n} X_i$
  - 显然, X是n次中取得正面的次数.
  - 由工具#1, 不难得出: $\mu \triangleq E[X] = \sum_{i=1}^{n} E[X_i] = \frac{n}{2}$

Markov说...

先用工具#2 (Markov不等式):

$Pr[X \geq t \cdot E[X]] \leq 1/t$

- 令$t = t'/E[X]$, 代入可得另一种形式: $Pr[X \geq t'] \leq E[X]/t'$
- 代入均值, 可得: $Pr[X \geq t'] \leq \frac{n}{2}/t'$
- 令$\frac{n}{2}/t' = 1/n$, 可得: $t' = n^2/2$

结论:

$$Pr\left[\text{至少有}n^2/2\text{次出现正面}\right] \leq \frac{1}{n}$$

Markov一定在发烧...

Chebyshev说…

- 再用工具#3 (chebyshev不等式): $Pr[|X-E[X]|>t\cdot\sigma]\leq 1/t^2$
  - 令$t=t'/\sigma$, 代入可得另一种形式:

$$Pr[X\geq t']=Pr\big[|X-E[X]|\geq t'-E[X]\big]\leq\frac{\sigma^2}{(t'-E[X])^2}$$

  - 由于$E[X_i]=1/2$, 可知:$\sigma^2(X_i)=E\big[(X_i-E[X_i])^2\big]=1/4$
  - 由于X_i相互独立, 故$\sigma^2(X)=\sum_{i=1}^{n}\sigma^2(X_i)=n/4$
  - 代入可得: $Pr[X\geq t']\leq\frac{n}{4\left(t'-\frac{n}{2}\right)^2}$

➡ 由此可解出$t'=n$

➡ 结论:

$$Pr[\text{至少有}n\text{次出现正面}]\leq\frac{1}{n}$$

➡ Chebyshev也在发烧, 比Markov温度稍低.

Chernoff说…

$$Pr[X > (1+\delta)\cdot E[X]] < \left(\frac{e}{1+\delta}\right)^{(1+\delta)E[X]}$$

再用工具#4 (Chernoff界):

- RHS可以进一步放大为$e^{-\frac{\delta^2}{3}E[X]}$.
- 代入均值可得: $Pr\big[X > (1+\delta)\cdot E[X]\big] < e^{-\frac{\delta^2}{6}n}$
- 令RHS = 1/n, 可得: $\delta = \sqrt{\frac{6\ln n}{n}}$

➡ 由此可得: $t = \frac{(1+\delta)n}{2} = \frac{n}{2} + \sqrt{\frac{3n\ln n}{n}}$

➡ 看起来合理多了:
令$n = 100$, 则 $t \approx 76$
令$n = 1000$, 则 $t \approx 602$

➡ Chernoff体温应该正常.

应用案例#2:哈希

- 我们来研究不同的哈希函数族有什么样的性能保障.
  - 考虑一族哈希函数$\boldsymbol{h} \in \mathcal{H}$, 每个函数完成的都是从数据空间$\boldsymbol{U}$到表项编号$\{\mathbf{1}, \mathbf{2}, \dots, \boldsymbol{n}\}$的映射.
  - 任给数据集$\boldsymbol{S} \subseteq \boldsymbol{U}$, 假定$|\boldsymbol{S}| = \boldsymbol{n}$. 随机挑选哈希函数$\boldsymbol{h} \in \mathcal{H}$, 用$\boldsymbol{h}$将$\boldsymbol{S}$中的数据映射到哈希表中.
  - 希望: $\boldsymbol{S}$中的元素被均匀散布在哈希表中.
    - 通常用表项“负载”来描述这种均匀程度.
    - 负载越小,均匀程度越高.
    - 若用链表来解决冲突, 则表项负载也决定了查找耗时.
- 对哈希函数的不同假设, 会导致不同的性能界.

H#1:第一个假设

- 均匀假设: (这是最弱的假设)
  - $\forall x \in U, \forall i \in \{1, \dots, n\}, Pr_{h\in\mathcal{H}}[h(x) = i] = 1/n$
- H#1有多弱?
  - 考虑常数哈希: $\mathcal{H} = \{h(x) = i : i = 1, 2, \dots, n\}$
  - 所有数据都被放入同一表项.
  - 但该函数族满足H#1.
- 但即使是在这种假设下, 仍可以得到看起来不错的结果.

应用工具#1

定义随机变量:

- 任给$\forall y \in S$, 令X_y表示数据y被映射到表项i的指示变量.

$$X_y = \begin{cases} 1, & h(y) = i \\ 0, & otherwise \end{cases}$$

- 令随机变量X表示表项i的负载.

➡ 显然: $X = \sum_{y \in S} X_y$

$E[X]$=?

➡ 由H#1可知, $E[X_y] = 1/n$

➡ 由线性期望, $E[X] = \sum_{y \in S} E[X_y] = 1$

虽然挺合理的, 但均值的内涵实在有限.

Markov说...

- 工具#2 (Markov不等式):
  - $Pr\big[X \geq t \cdot E[X]\big] \leq 1/t$
  - 令$t = n$, 有: $Pr[\text{表项}i\text{的负载} \geq n] \leq 1/n$
  - 解读: 10个数据全部映射到一个表项里的概率不超过1/10.
  - 如果你对哈希的理解足够, 这种性能保证很难令人满意.
  - 但在只有**H#1**这么弱的假设下, 你只能得到这样的保证.
  - ➡ 那族常数哈希的确达到了这个界.
- 哈希函数在应用中, 除了需要满足均匀假设外, 还需要满足独立性假设.

H#2:第二个假设

成对独立假设:

- $\forall x, y \in U, x \neq y, \forall i, j \in \{1, \ldots, n\},$
  $$Pr_{h \in \mathcal{H}}[h(x) = i, h(y) = j] = \frac{1}{n^2}$$
- 换句话说, 该假设下, 任给一对数据, 哈希函数的决定就像完全独立、且完全均匀的一样.[联合概率被分解]
- 但是, 成对独立不是完全独立.
  [X1表示抛出硬币正面, X2表示另一个硬币抛出正面, X3=X1+X2]
- 满足H#2的函数族通常称为pairwise哈希.
- 在理论上大致相当于Universal哈希的要求.[实际中等效]
- 常数哈希显然不满足这个假设.

H#2意味着什么?

- 该假设意味着X的方差可以估计, 而且较小.
  - $Var[X] = \sum_{y \in S} Var[X_y]$
    - 一般来说,该式不成立. [X1和X2代表两个互补事件, 则X1+X2永远为1, 方差为0]
    - 事实上, $Var[X]$展开后,除了方差之和外, 还有一堆协方差项.
    - 但在H#2下, 所有协方差为0.
  - 由于 $Var[X_y] = \frac{1}{n}\left(1-\frac{1}{n}\right)^2 + \left(1-\frac{1}{n}\right)\left(-\frac{1}{n}\right)^2 = \frac{1}{n}\left(1-\frac{1}{n}\right)$
  - 故有:$Var[X] = n \cdot \frac{1}{n}\left(1-\frac{1}{n}\right) \leq 1$
    - 作为对比, 常数哈希下, $Var[X] = n$

方差有界意味着什么?

$Pr[|X - E[X]| > t \cdot \sigma] \leq 1/t^2$

- 可以听Chebyshev怎么说了.
  - 令$t = n$, 有: $Pr[\text{表项}i\text{的负载} \geq n] \approx Pr[|X - 1| \geq n] \leq 1/n^2$
  - 解读: 差不多有1%的概率, 会把10个数据全部映射到同一表项.
    - 这无疑比只有H#1的情况下得到的性能保证要好得多.
    - 但其实仍然不够好.
- 正如你可能已经预料到的, 如果加强独立性假设, 可望得到更好的性能保证.

H#3:第三个假设

- 完全独立假设:
  - 所有的h(x)都**均匀且独立**地散布于$\{1,2,\ldots,n\}$中.
    - 这是最完美的哈希函数.
    - 完美到不可实现. [我们生活在一个不可能存在完美的世界]
    - 这差不多等价于随机哈希函数.[每来一个数据都随机选择一次]
    - 显然, 这不可能.
    - 现实中满足H2的函数族存在; 满足H3的没有;
      但介于二者之间的那些函数族正是人们追求的目标.
- 仍然值得追问: 这种完美假设下, 能达到怎样的性能保证.

该听听Chernoff了...

- **H#3下, X是由n个独立的0-1变量求和得到的.**
  - 应用Chernoff界(part a), 令$1+\delta=\ln n$, 可得:
    $$Pr[X \geq \ln n] < \left(\frac{e}{\ln n}\right)^{\ln n}$$
  - 解读: 首先注意到$\left(\frac{1}{e}\right)^{\ln n}=\frac{1}{n}$.
    可推广到: $a^{\ln n} \approx \frac{1}{n^d}, 0 < a < 1$, 其中a越小,d越大.
  - 再注意到我们的结论中, e/ln$n$可以任意小.
  - ➡ 这个概率小于任意的“多项式倒数”函数. 即:WHP, $X \leq \ln n$
  - 对比Markov和Chebyshev推得的结论.
  - ➡ 偏移更小. [ln$n$ vs. n]
  - ➡ 概率也更小. [可以小于任意的多项式倒数]

类似的另一个结论

- 同样基于Chernoff界, 换种推导方式, 可得:

  - $$Pr\left[X \geq \frac{3\ln n}{\ln\ln n}\right] \leq \frac{1}{n^2}$$

  - 这种形式在算法分析中也经常看到.

  - 我们将来还会看到.

- 至此, 我们讨论的都是“单个表项i的负载”.

- 更有意义的答案通常是: 整张表(n个表项)的负载上界.

应用工具#5

定义:

- $\forall \boldsymbol{i} \in \{\mathbf{1}, \mathbf{2}, \dots, \boldsymbol{n}\}$, 令$\boldsymbol{E_i}$表示事件: 表项$\boldsymbol{i}$的负载超过$\frac{\mathbf{3}\ln \boldsymbol{n}}{\ln\ln \boldsymbol{n}}$

根据Union Bound, 可知:

$$\boldsymbol{Pr}[\text{至少有一个事件发生}] \le \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} \Pr[\boldsymbol{E_i}] \le \frac{\mathbf{1}}{\boldsymbol{n}}$$

换句话说, "没有任何表项负载超过$\frac{\mathbf{3}\ln \boldsymbol{n}}{\ln\ln \boldsymbol{n}}$" 的概率超过$\mathbf{1} - \frac{\mathbf{1}}{\boldsymbol{n}}$.

经常也表述为: WHP, 最大负载小于$\mathbf{O}\left(\frac{\ln \boldsymbol{n}}{\ln\ln \boldsymbol{n}}\right)$.

小结: 5个工具

- 工具#1: 线性期望
  - 如果你关心的只是均值, 这个工具通常就够了.
- 工具#2: Markov不等式
  - 如果你只需证得常数界, 这个工具通常足够了.
- 工具#3: Chebyshev不等式
  - 如果你能有效控制方差, 那么这个工具得到的结果比Markov好.
- 工具#4: Chernoff界
  - 如果随机变量是多个独立有界的随机变量之和, 一定要试试这个.
- 工具#5: Union Bound
  - 可以用来避免多个低概的坏事件.

其中四个都非常简单, 五个都非常有用.

Rounding

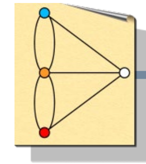
1. 5 Tools for prob. analysis
2. Min-Congestion Routing

概要

让我们回到近似算法.

- 本节将应用上述概率分析工具来讨论一个具体的随机取整算法.
  - 具体来说, 会用到工具#1, #4和#5.
- 该算法针对的是最小拥塞路由问题.[RR的典型应用]
- 该问题在实质上与LP建模那部分讨论过的分离路径问题是一致的,
  - 建模成带有优化目标的模型更便于陈述和理解.

最小拥塞路由问题



Edge-ILP

$$\text{Minimize } t$$

subject to

$$\sum_{e\in\delta^+(d_i)} x_{e,i} = \sum_{e\in\delta^-(s_i)} x_{e,i} = 1, \qquad \forall i$$

$$\sum_{e\in\delta^+(v)} x_{e,i} = \sum_{e\in\delta^-(v)} x_{e,i}, \forall i, \forall v \neq s_i, d_i$$

$$\sum_{i=1}^{K} x_{e,i} \leq t, \qquad \forall e \in E$$

$$x_{e,i} \in \{0,1\}, \qquad \forall i, \forall e \in E$$

- K对顶点, $(s_1, d_1), \ldots, (s_K, d_K)$
- 变量定义在边上.
- $x_{e,i}$:第i对s-d的路径是否经过e.
- $\delta^+(v)$:顶点v的入度边.
- $\delta^-(v)$:顶点v的出度边.

该问题在求解什么?

→ 为那K对顶点各安排一条路, 希望这K条路“重用”的边数目最小化.

→ 0-1约束改为非负约束,得到**Edge-LPR**.

该问题是NP-H问题. 通信网、电路设计等领域用途广泛.

Path-ILP

将变量定义在路径上, 可得等价的模型.

Path-ILP

$$\text{Minimize } t$$

subject to

$$\sum_{P\in\mathcal{P}_i} x_P = 1, \qquad \forall i$$

$$\sum_{i=1}^{K}\sum_{e\in P, P\in\mathcal{P}_i} x_P \le t, \forall e\in E$$

$$t \ge 0,$$

$$x_P \in \{0,1\}, \qquad \forall P$$

- $\mathcal{P}_i$: 第i对s-d之间存在的所有路径.
- x_P: 是否使用了路径*P*.

➡ 0-1约束改为非负约束,得到Path-LPR.

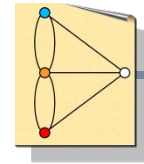
为什么需要两个等价的模型?

➡ 取整的依据是Path-LPR的解.

➡ 但算法第一步求解的是Edge-LPR.
[因为Path-LPR的变量数目为指数]

如何变换?

Flow分解算法



- 输入Edge-LPR的解$\{x_{e,i}\}$, 输出Path-LPR的解$\{x_P\}$.
  - 从s_i开始,跟踪$x_{e,i} > 0$的边, (有多个选择时,任选一个), 直至到达d_i.
    - 得到一条路径*P*.
  - 找到路径*P*上最小的$x_{e,i}$; 从*P*的每个边上都减去该最小值.
  - 记录x_P为该最小值.
  - 重复以上三步, 直至s_i的流量为0.
  - 对其他的s-d对重复以上步骤.
- 可以证明, 这样得到的$\{x_P\}$满足以下性质:
  - 是Path-LPR的可行解.
  - 目标值为FOPT, 与$\{x_{e,i}\}$一样.
  - $\forall i, \sum_{P \in \mathcal{P}_i, e \in P} x_P = x_{e,i}$

随机取整算法

算法:

- **Step#1:** 调用**LP-Solver**求解**Edge-LPR**, 得到$\{x^*_{e,i}\}$.
- **Step#2:** 调用流分解算法, 得到$\{x^*_P\}$.
- **Step#3:** 将$\{x^*_P\}$理解为概率, 随机取整.

取整方法跟以前一样.

- 假定针对第**i**对**s-d**, $\{x^*_P\}$中用了三条路, 流值分别为**0.5**, **0.3**和**0.2**
- 产生**[0,1]**内均匀分布的随机数$\boldsymbol{\beta}$.

  若$\boldsymbol{\beta} \in [0, 0.5]$,则选择第一条路; 若$\boldsymbol{\beta} \in [0.5, 0.8]$, 则选第二条路; 若$\boldsymbol{\beta} \in [0.8, 1]$,则选第三条路.

- 关键点: 针对每个**s-d**对, 随机选择都是完全独立的.

分析:均值

拥塞程度的均值:

- 令$X_{e,i}$表示最终的解中, 第i对s-d的路径是否经过边e的指示变量.
  - 很明显, $Pr\left[X_{e,i}\right] = x_{e,i}^{*}$
  - 因此, $E\left[X_{e,i}\right] = x_{e,i}^{*}$
- 令X_e表示随机变量: 取整后有多少条路径经过了边e.
  - 显然, $X_e = \sum_{i=1}^{K} X_{e,i}$
  - 由线性期望可知, $E[X_e] = \sum_{i=1}^{K} E\left[X_{e,i}\right] = \sum_{i=1}^{K} x_{e,i}^{*} \leq t^{*} = FOPT$
- 换句话说, 平均来看, 边上的拥塞程度最多为t^{*}.

为了得到更可靠的结论, 需要保证尾部概率足够小.

- X_e是一组取值为0-1的独立随机变量之和, 满足Chernoff界的条件.

又来做梦...

推导的目标:

- 假如我们能证得, 存在λ, 使得$\forall e \in E$, 都有(其中$m = |E|$):

$$Pr[X_e \geq \lambda t^*] \leq \frac{1}{m^2}$$

➡ 根据Union Bound可知:

$$Pr[\exists e, X_e \geq \lambda t^*] \leq \frac{1}{m}$$

➡ 可表述为: WHP, 算法解比FOPT最多差λ倍.

➡ 我们得到的是个λ -近似算法.

应用Chernoff界

推导:

- 令$\mu = E[X_e]$, 前面讨论过, $\mu \le t^* = FOPT \le OPT$
- 我们有:

$$Pr[X_e > \lambda \cdot OPT] \le Pr[X_e > \lambda\mu] \le \left(\frac{e^\lambda}{\lambda^\lambda}\right)^\mu$$

$\Rightarrow$ $\mu \le OPT$

$\Rightarrow$ 令$\lambda = (1+\delta)$, 套Chernoff界

- 无论μ取值如何,均可证明, 当$\lambda = O(\frac{\ln m}{\ln\ln m})$时, $\left(\frac{e}{\lambda}\right)^{\lambda u} \le \frac{1}{m^2}$
- 出现如此古怪的函数形式, 是因为方程$x^x = m$的解大致为$\frac{\ln m}{\ln\ln m}$.

Claim-13: 这个随机取整算法$O(\frac{\ln m}{\ln\ln m})$ -近似.

小结

- 五个工具: 随机近似算法的分析中非常有用
- 最小拥塞路径: 随机取整技术最典型的应用
  - **Edge-ILP vs. Path-ILP; Flow Decomposition**
  - 分数解与概率解读;
  - **WHP**概念的应用;
  - **Chernoff**界的应用.

基于LP的近似算法

1 Set Cover

2 Min. Congestion : Rounding

3 MultiCut : Primal-Dual

Multicut: 背景

在单商品流问题中:

- 最小s-t割问题及其对偶(最大s-t流)占据了核心地位.
- 围绕这一对问题, LP理论和各种组合优化方法被联系在一起.

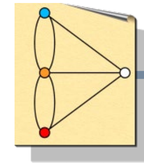
在多商品流问题中:

- 同样存在着地位类似的一对问题:
  最小Multicut vs. 整数最大多商品流

➡ 前面讨论过的最小拥塞路由问题是后者的一种特殊变形.

➡ 我们下面要讨论的算法针对的是前者的一种特例.

最小Multicut问题



问题描述:

- 给定无向图$G=(V,E)$
- $\forall e\in E$, 给定边上容量c_e.
- 给定K对顶点, $\{(s_1,t_1),(s_2,t_2),\ldots,(s_K,t_K)\}$
  - Pair之间各不相同, 但可以有相同的顶点.
- **Multicut**是一个边的子集, 去掉这些边, 会使得所有的**s-t**对被分开.
- 该问题的目标: 最小容量的**multicut**.

Notes:

- 若**K=1**, 退化为最小**s-t**割问题. 多项式可解.
- 对任意**K**值, 该问题**NP-H**.

Tree-Multicut问题

- 我们这里关注的是最小Multicut问题的一个特例.
  - 给定的图是一棵无向树; 其他问题描述不变.
    - 由于是树, 任意一对顶点间只有唯一路径.
    - 令(s_i, t_i)之间的那条路为p_i.
    - 为了分割(s_i, t_i), multicut中至少要包含p_i上的一条边.
    - 或许你觉得这个问题容易多了, 但它仍然是NP-H. [Vertex Cover问题可以归约为该问题]
- 本节任务:
  - 发展一个针对**TM**问题的**P-D**算法.
    - 因此需要先给出TM-ILP, TM-LPR, 和TM-LPR-D.

ILP vs. LPR

TM-ILP

$$\text{Minimize} \sum_{e \in E} c_e d_e$$

subject to

$$\sum_{e \in p_i} d_e \geq 1, \forall i \in \{1, \ldots, K\}$$

$$d_e \in \{0,1\}, \forall e \in E$$

TM-LPR

$$\text{Minimize} \sum_{e \in E} c_e d_e$$

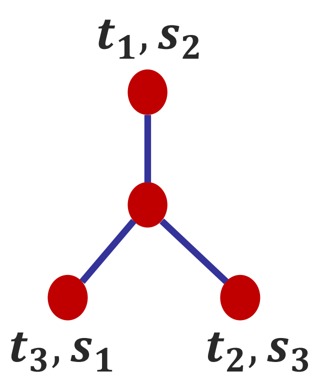
subject to

$$\sum_{e \in p_i} d_e \geq 1, \forall i \in \{1, \ldots, K\}$$

$$d_e \geq 0, \forall e \in E$$

Notes:

- d_e为决策变量.
- 同样, LPR中不需要约束$d_e \leq 1$
- LPR的解称为分数multicut: $\forall p_i$, 其上各边的取值之和超过1.
- 很明显, FOPT≤OPT.
- 考虑下例.



$\forall e, c_e = 1$

➡ 为了分割三对顶点, 至少需要2条边. OPT=2

➡ 考虑分数解:$\forall e, d_e = \frac{1}{2}$
每条路含两条边, 故满足约束. FOPT=3/2

LPR vs. LPR-D

对偶及其含义:

- 经标准推导可得TM-LPR-D.
- 容易发现, 这是个多商品流模型.

➡ f_i表示路径p_i上的流.

➡ 该问题要求在K对顶点之间搬运流, 希望总流量最大化, 但必须满足边上的容量约束.

➡ 唯一的区别是: 这里是无向图, 因此计算一条边上的总流量时, 无论哪个方向经过该边, 都需要累加.

➡ 这个有明确物理意义的问题, 称为"树上最大多商品流".(Tree-Flow)

TM-LPR

$$\text{Minimize} \sum_{e\in E} c_e d_e$$

subject to

$$\sum_{e\in p_i} d_e \geq 1, \forall i \in \{1, \dots, K\}$$

$$d_e \geq 0, \forall e \in E$$

TM-LPR-D

$$\text{Maximize} \sum_{i=1}^{K} f_i$$

subject to

$$\sum_{i: e\in p_i} f_i \leq c_e, \forall e \in E$$

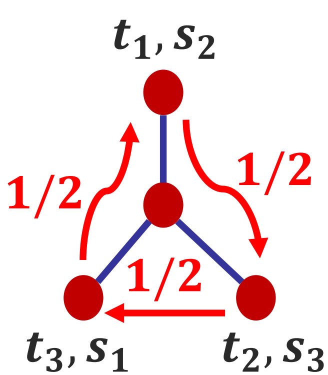
$$f_i \geq 0, \forall i \in \{1, \dots, K\}$$

TF-LPR vs. TF-ILP

更进一步:

- 上述TF问题的解显然是分数解.
  - 若将非负约束改为0-1整数约束, 则得到的是相应的整数TF问题.
    [当然, 其他问题输入也必须是整数]

TF-ILP vs. TF-LPR



$\forall e, c_e = 1$

- 显然, 分数最佳解为: 三条路径都是1/2. 总和为3/2.
- 而整数最佳解为: 只能选一条路, 流为1.
- 毫不意外: FOPT≥OPT

TF-ILP

$$\text{Maximize} \sum_{i=1}^{K} f_i$$

subject to

$$\sum_{i:e\in p_i} f_i \le c_e, \forall e \in E$$

$$f_i \in \{0,1\}, \forall i \in \{1,\dots,K\}$$

TF-LPR(TM-LPR-D)

$$\text{Maximize} \sum_{i=1}^{K} f_i$$

subject to

$$\sum_{i:e\in p_i} f_i \le c_e, \forall e \in E$$

$$f_i \ge 0, \forall i \in \{1,\dots,K\}$$

Multicut vs. 多商品流

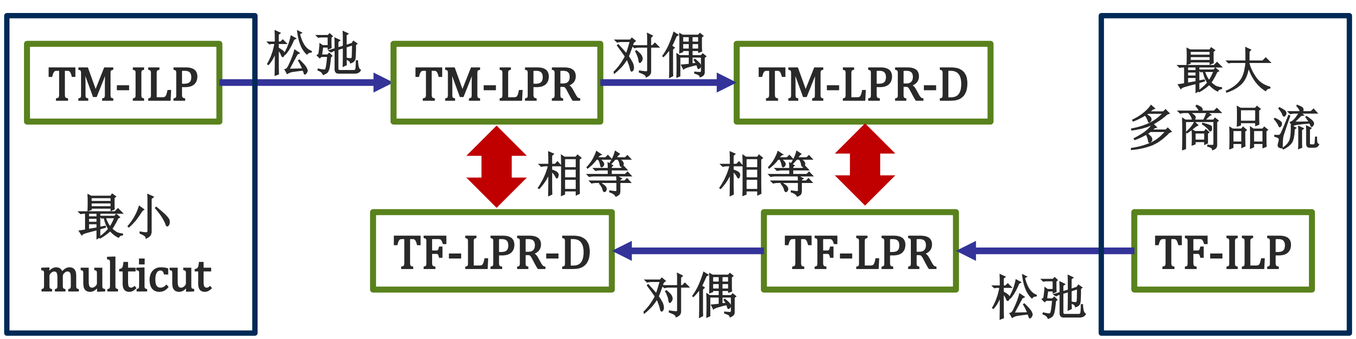
再进一步:

- 假如给定的不是树, 而是一般图; 其他描述与TF-ILP一致.

最大多商品整数流问题: 图G, s-t对都与最小multicut问题一样, 但边上容量为整数. 对每个s-t对定义一种商品. 希望运送的商品总量最大化. 要求满足边上的容量约束, 且流量为整数.

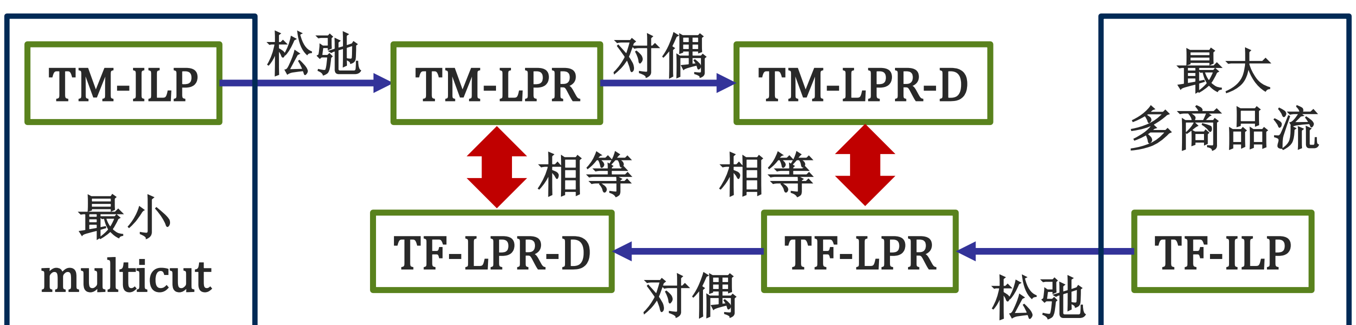
TF显然是该问题的特例.

小结一下:



P-D算法干了什么?

- 接下来讨论TM-ILP问题的Primal-Dual算法.
  - A) 维护一组主解, 一组对偶解.
  - B) 始终保持对偶可行, 追求主可行.
  - C) 始终保证主变量互补松弛条件满足.
  - D) 始终保证对偶互补松弛条件近似满足.
- 从TF问题的角度看, 该算法干了些啥?
  - 把上面四句话中的“主”和“对偶”交换一下.
  - 该算法同时求解了TF问题. 且近似比一样(2 vs. ½)



P-D算法概要设计

让我们对着上述四条特征来看看如何实现.

- A) 维护一组主解, 一组对偶解.
  - 这个容易. 设置两组变量而已. 初始化时, 流*f*为0; 割边**D**为空集.
- B) 始终保持对偶可行, 追求主可行.
  - 也不难. 每次增大*f*时, 都保证容量约束满足. **D**集合添加边的方式保证主变量为整数; 直至最后**D**集合可行: 满足multicut的条件.
- C) 始终保证主变量互补松弛条件满足.
  - 同样不难. 只需保证每次向**D**中加边时, 这些边上的容量被用满(对偶约束变紧). 算法中正是基于这个条件来决定加入哪条边.
- D) 始终保证对偶互补松弛条件近似满足.
  - 这个麻烦了.

设计难点

- 如何保证对偶互补松弛条件近似满足.
  - 在VC问题中, 这并不难, 按照C定义的方式选择顶点, 自动就保证了最多超过主约束2倍.

但在这里如何保证选出来的边的数目有界(即$\sum_{e \in p_i} d_e \le \beta$)?

- 该算法做了两件事:
  - 流选择方式: 每次增加流f时, 都用一种特殊的方式来决定增加那些路径流.
  - 删减步骤: 在得到主可行解后, 还对该解进行了一次删减(去掉D中冗余的边)

➡ 最终保证了$\beta = 2$.

➡ 从而得到2-近似算法.

TM-LPR

$$\text{Minimize} \sum_{e \in E} c_e d_e$$

subject to

$$\sum_{e \in p_i} d_e \ge 1, \forall i \in \{1, \dots, K\}$$

$$d_e \ge 0, \forall e \in E$$

几个术语

顶点深度(depth):

- 从顶点v出发到达根顶点的路径长度(跳数), 称为v的深度.
  [根顶点任意选定; 显然, 根的深度为0]

最低公共祖先(lowest common ancestor, lca)

- 任给两个顶点$u, v \in V$, $lca(u, v)$ =u-v路径上深度最小的顶点.

“更深”的含义(deeper)

- 考虑一条从顶点v到根的路径.
- 从v开始“上行”, 先碰到的点(或边)比后碰到的点(或边)更深.

P-D算法for TM

TM-P-D

- Step#1(初始化): $D = \Phi, f = 0$
- Step#2(Flow Routing):
  - 按深度降序, 逐一检查图中顶点v
    - $\forall(s_i, t_i), s.t.\ lca(s_i, t_i) = v$, 贪心增加$p_i$上的流量$f_i$, 直至有边被耗尽.
    - 将所有耗尽边加入D.
- Step#3(Reverse Deletion): [令$e_1, \dots, e_l$为D中边的加入顺序]
  - For $j = l$ downto 1
    - IF $D - \{e_j\}$仍是multicut, THEN $D = D - \{e_j\}$
- Step#4: 输出D和f

流选择方式:
1) 由顶点来决定哪些流增大;
2) 顶点按照深度排序.

删减步骤:
1) 去掉“冗余”;
2) “逆序”删减.

为什么这两个设计可以保证近似比?

关键引理

Lemma-14: 令(s_i, t_i)得到了非零流量, 且$lca(s_i, t_i) = v$. 则:
(a) s_i到v的路径上最多只有一条边被选入D.
(b) v到t_i的路径上最多只有一条边被选入D.

- 我们只证明结论a, 结论b同理可证.
- 假定$s_i - v$路径上有两条边e和e'被选入D; WLOG, e比e'更深.
- 考虑逆序删减过程中, 检查到e的那个时候.

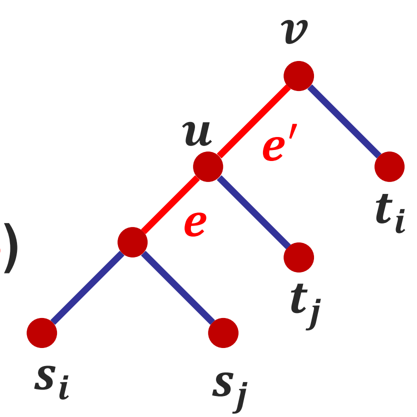
什么原因使得e没有被删除?

- 一定是因为除了(s_i, t_i)之外, 还有另一对(s_j, t_j)依赖于e才能分隔开.
- 换句话说, 检查e时D中只有e在路径p_j上. (**事实#1**)
- 这同样也说明: e'不在路径p_j上.

证明(续)

Lemma-14: 令(s_i, t_i)得到了非零流量, 且$lca(s_i, t_i) = v$. 则:
(a) s_i到v的路径上最多只有一条边被选入D.
(b) v到t_i的路径上最多只有一条边被选入D.

- 令$lca(s_j, t_j) = u$
  - ➡ “e'不在路径p_j上”足以说明: u比e'更深.
  - ➡ 所以, u比v更深. ➔ 算法会先处理u.(事实#2)
- 处理u时, 路径p_j上至少一条边(令为e'')加入D.(事实#3)



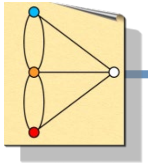
? e是什么时候加入D的?
  - ➡ 处理v时, 或者在那之后. 否则p_i上流量为0.(事实#4)
- 由事实#2、#3和#4可知, e''比e更早加入D.
- 逆序删除意味着, 检查e时, e''应该还在D中.
  - ➡ 这与事实#1矛盾.
- 得证.

可行性

Claim-15: 对树上最小Multicut问题, 上述P-D算法2-近似. 对树上最大多商品整数流问题, 上述P-D算法1/2-近似.

- 先证明可行性.
  - Step#2结束时, *f*是整数极大流.
  - 因此, *f*是可行的.
  - Step#2结束时的D必然是Multicut(否则*f*不可能是极大流)
  - Step#3删除的边是冗余的.
  - 因此, 最后输出的**D**可行.

近似比



Claim-15: 对树上最小Multicut问题, 上述P-D算法2-近似. 对树上最大多商品整数流问题, 上述P-D算法1/2-近似.

- 再看近似比.

  - D中每条边都是容量耗尽的. ➔ 主互补松弛条件满足.
  - 由Lemma-14, 非零流路径上, 最多只有两条边被加入D.
  - 因此, 对偶互补松弛条件“2倍近似满足”. [D中边的数目最多为流量*f*的2倍]
  - 由于可行流是最佳Multicut的下界.
    ➔该算法对Multicut问题2-近似.
  - 由于可行Multicut是最佳多商品流的上界.
    ➔该算法对多商品流问题1/2-近似.

- 得证.

小结

- 由于对偶关系的存在, **Primal-Dual**算法往往可以同时求解两个问题.
- 对偶互补松弛条件的“近似满足程度”如何定界, 往往是算法设计的难点.
- 根据具体问题, 巧妙设计对偶解的改进顺序, 是克服该困难的有效手段.
- 对得到的主可行解进行“裁减”, 同样是有效手段.
- 至少对某些问题, 裁减的顺序很关键.