NP Polunomial-Time Reduction

If $X \le_p y$ and y can be solved in polynomial-time, then X can be solved in polynomial time

If $X \le_p y$ and X cannot be solved in polynomial-time, then y cannot be solved in polynomial time

If $X \le_p y$ and $y \le_p X$, we use notation $X \equiv_p y$. In this case, X can be solved in polynomial time iff y can be

Independent Set 独立集

在图中一个节点集合S中没有两个节点被边连接S就是一个独立集，最小独立集简单一个节点就是，困难在于最大独立集。独立集问题：给定图G和数k，G是否包含大小至少为k的独立集。

Vertex Cover 顶点覆盖

在图中每条边e在节点集合S中至少有一个是其端点，则S是一个顶点覆盖，最大顶点覆盖简单所有节点既是，困难最小顶点覆盖。顶点覆盖问题：给定图G和数k，G是否包含大小为k的顶点覆盖

Vertex Cover = Ind. Set

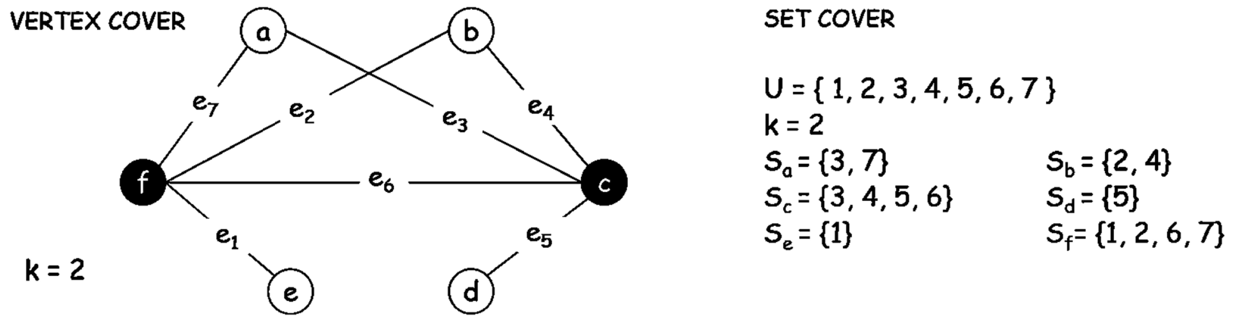
图G(V,E)如果S是一个独立集，那么任意边e(u,v)由于S是独立的，u v不可能同时出现在S中，那么必有一个端点在V–S中，则V–S是一个顶点覆盖。反方向，如果V–S是一个顶点覆盖，考虑S中任意两个节点u v，如果u v之间有边那么u v必不同时在V–S中，但是V–S是一个顶点覆盖，矛盾，这S中没有两个节点有边连接，S是一个独立集。规约时可以问对方是否存在n–k

Set Cover 集合覆盖

给定n个元素集合U，以及m个U的子集Sm和数字k，是否存在这些集合中至多k个集合其并集为U。如m个软件，期望选择n个功能使用最少的几个软件。

Vertex 规约到 Set Cover

给定一个图G，我们将顶点看作子集，元素为与其连接的边序号，构建多个子集，之后在这些集合上运行set cover算法



集合打包问题：给定n个元素的集合U，子集数字k，是否存在至少k个集合两两不相交。使用独立集问题规约到set打包问题，构造方法与上面的方法相同。

SAT 与 3-SAT问题

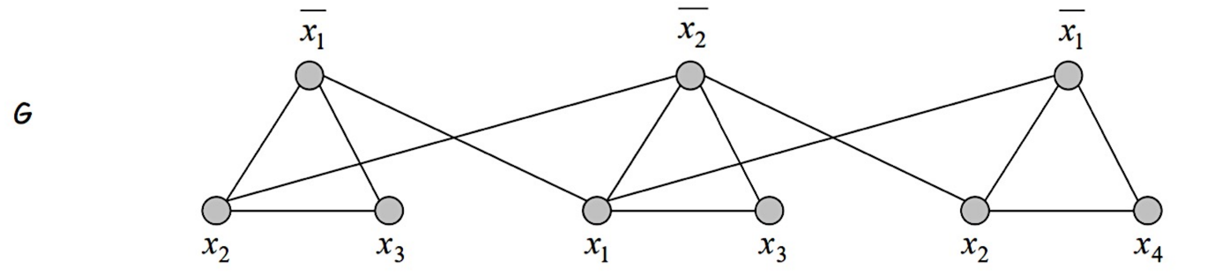
SAT. Given CNF formula Φ, does it have a satisfying truth assignment?

•3-SAT: SAT where each clause contains exactly 3 literals

Ex: $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

3SAT 到 独立集

针对每一个子句，构建一个三角形端点为变量，之后将变量与其他三角形中相反的变量连接起来。



k = 3

$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Claim. G contains independent set of size k = |**Φ**| iff **Φ** is satisfiable

- **Pf.** Let S be independent set of size k
- S must contain exactly one vertex in each triangle
- Set these literals to true ← and any other variables in a consistent way • Truth assignment is consistent and all clauses are satisfied
- **Pf.** Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k

3–SAT 到 独立集 到 顶点覆盖 到 集合覆盖

P 问题：可以在多项式时间内解决，判定的问题

NP 问题

针对一个问题，有一个证书certifier or witness， NP 问题时一个 decision problem 存在一个多项式时间的 certifier，如果有证书可以多项式时间验证真假。

其他问题

P Decision problems for there is a **poly-time algorithm**
EXP. Decision problems for which there is an **exponential-time algorithm** **NP** Decision problems for which there is a **poly-time certifier P 属于 NP，NP 属于EXP。**

NPC问题

NP-complete. A problem Y in NP with the property that for every problem X in NP, X $\leq_P$ Y

第一个NPC问题 电路 Circuit-SAT

3 SAT 是 NPC，可以将电路规约到 3SAT

$X_2 = \neg X_3$ add 2

$X_1 = X_4 \vee X_5$ add 3

$X_0 = X_1 \wedge X_2$ add 3

$$x_2 \vee x_3 \,,\quad \overline{x_2} \vee \overline{x_3}$$

$$x_1 \vee \overline{x_4},\quad x_1 \vee \overline{x_5} \,,\quad \overline{x_1} \vee x_4 \vee x_5$$

$$\overline{x_0} \vee x_1,\quad \overline{x_0} \vee x_2,\quad x_0 \vee \overline{x_1} \vee \overline{x_2}$$

Hard-coded input values and output value

X_5 = 0add 1 X5 的非

X_0 = 1add 1 X0

证明NPC的四个步骤

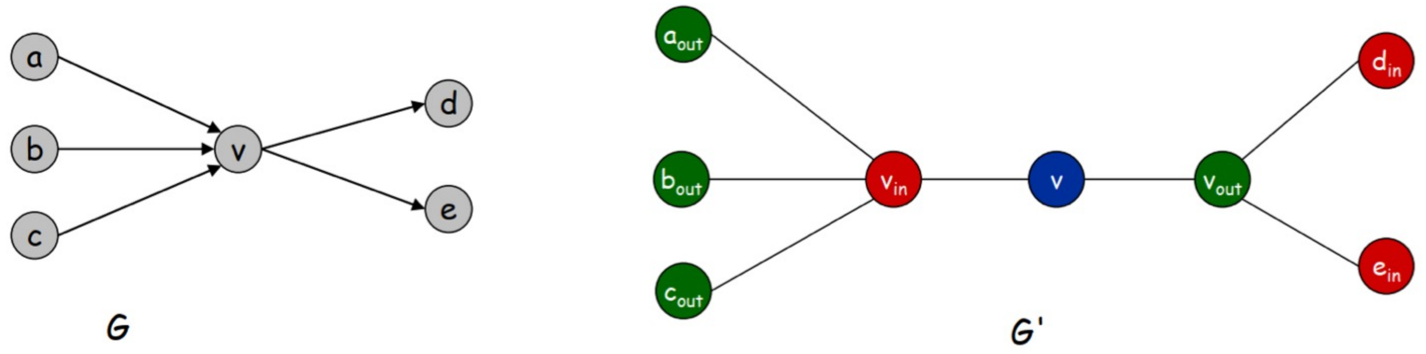
1 说明这个问题是一个NP问题，可以在多项式时间验证结果
2 找一个已知的NPC问题，**规约到**现在的未知问题上（注意方向）一般是将已知的NPC的问题的数据结构变化出一个新数据，之后feed到新问题中求解。
3 说明这个规约步骤是多项式时间内规约上去的。
4 说明如果NPC问题有解，则被证问题有解，如果被证问题有解则NPC问题有解，相当于充要条件。整体上重要的是，我们的证明是证明新问题至少与已知NPC问题难度相同，如果新问题更难，解决问题更广，也是可以的。

NP 和 co-NP

Co–NP是NP的反问题，如果NP不等于co–NP那么P不等于NP

Hamiltonian Cycle 图中的一条路，经过每个顶点恰好一次

Claim. DIR-HAM-CYCLE $\leq_P$ HAM-CYCLE **Pf** Given a directed graph G = (V, E), construct undirected graph G′ with 3n nodes



Claim G has a Hamiltonian cycle iff G′ does
Pf ->Suppose G has a directed Hamiltonian cycle Then G′ has an undirected Hamiltonian cycle (same order) **Pf←**
Suppose G′ has an undirected Hamiltonian cycle
The cycle must visit nodes in G′ using one of following two orders: • …,B,G,R,B,G,R,B,G,R,B,… • …,B,R,G,B,R,G,B,R,G,B,…
Blue nodes in this cycle make up directed Hamiltonian cycle in G, or reverse of one

TSP 问题

考虑一个必须访问n个城市的推销员，推销员从家v1出发，希望找到一条路径，可以访问所有城市且回家，任意两个城市之间距离已经给出，要求找到一条总距离最近的路径。
HAM–CYCLE 可以规约到 TSP，TSP比HAM更难，在HAM图中创建n个城市，两个城市之间如果有边，定义距离为1,如果没有定义距离为2,TSP存在一个tour距离小于n iff G 是一个 HAM

Longest 问题

Shortest–Path 一个图中是否有一个简单路径，路径长度最长为k条边。 **Longest–Path** 图中是否存在一个简单路径长度最少为k条边

3D Matching 三维匹配

Given n instructors, n courses, and n times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

Given disjoint sets X,Y and Z, each of size n and a set T⊆X×Y×Zof triples, does there exist a set of n triples in T such that each element of X ∪ Y ∪ Z is in exactly one of these triples?

3-Colorability 三着色问题

给定一幅图，是否存在一种方法对端点着色，使得临近端点颜色不同，三着色指用三种颜色。这种问题是常见的，在资源冲突问题中我们可以考虑使用三着色问题分析。假设在系统上有n个进程，系统有k个时间槽或者k个时间段，希望安排进程，不希望出现冲突，这就可以在G上构建，如果两个进程有冲突用边连接，如果有k着色，则表示无冲突。

SUBSET-SUM 问题 与 背包问题 KNAPSACK

与数字相关的一般与子集和问题相关，作业3-1中，是否存在子集和等于整体和的一半，S/2，那么可以从子集和问题规约。

A set X and a target number t the subset sum is find a subset Y, so that the sum of Y is equal to t. Define the sum of X is s. Creat a item s-2t, add to the X as the new set S, feed the new set S. New S sum 2s-2t

CLOSESAT 指接受n-1个子句，只需要加入一个新变量并且加入原来的f，变为f交x交x的非，n-1必然包含x或x的非。**Schedule courses** 指寻找一个时间安排，k个时间槽，使用k着色规约，颜色代表时间槽进行规约。**Pairwise disjoint** 指选择k个子集合，这些子集合不相交，使用独立集规约，依旧，端点作为集合，元素为其连接的边。**命中集问题** 指有集合A与A的一组子集B如果有集合H包含来自每个Bi集合的至少一个元素，那么称H命中所有集合B，问题有给定一个集合A与其子集和数字k，是否存在一个命中集H大小最多为k，使用vertex cover规约，每一条边放在集合中，顶点作为子集合，FPT中算法可以是选择子集合，recursively测试xi has a k-1 hitting set运行时间满足T(m,k)<=cT(m,k-1)+O(cm)结果O(c^k*kcm) **多项式时间算法如interval schedualing** 算法是可以规约到NPC问题的，如果P=NP，NPC问题也可以规约到P问题上去**差异化子集问题** 客户的子集S中如不存在两个客户选择相同的商品，那么S是一个差异化子集，问是否存在一个大小为k的差异化子集，使用独立集规约，依旧客户作为节点，产品作为边，显然找到一个独立集说明有一个差异化子集，**高效招聘问题**n项运动m个人，可以将运动作为边，人节点，使用vertex cover规约。**资源分配问题**有n个进程和m个资源，如何分配使得至少有k个进程活跃，使用独立集规约，问题相当于找k个进程，访问的资源作为边不相接disjoin。不同的是k=2可以暴力n方求解，如果是特定人对特定资源变为一个二分图问题，如果一个资源可以被两个访问，依旧NPC。**单调SAT问题**如果变量只有x没有x非，全赋1就满足，但是找最小是一个NPC的，最小使用vertex cover解决，一条边xy对应一个子句x或y。**四维匹配**可以使用三维匹配规约，从三维匹配开始，加入z变成四元。**冰箱贴字母单词**这个问题比三维匹配难，从三维匹配规约三个集合，尝试找一个匹配只用一次**路径选择**选择k路径不共享任何节点使用三维匹配规约三个点

Extending tractability or FPT fixed parameter tractable

Solve some special cases of NP-complete problems that arise in practice.

Finding Small Vertex Covers

Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $S \le k$, and for each edge u,v,either $u \in S$, or $v \in S$, or both

如果k很小，那么暴力方法可以在O(kn^k+1)时间下完成，如果k是一个参数，那么这个算法是多项时间的。FPT期望找到一些参数k，让这个问题可以在f(k)–poly(n)

The following algorithm determines if G has a vertex cover of size $\le k$ in $O(2^k n)$ time

```
Vertex-Cover(G, k) {
   if (G contains no edges)    return true
   if (G contains ≥ kn edges) return false

   let (u, v) be any edge of G
   a = Vertex-Cover(G - {u}, k-1)
   b = Vertex-Cover(G - {v}, k-1)
   return a or b
}
```

解决树上的NP难问题

首先是独立集问题，最普通的独立集问题我们可以使用贪心方法解决，如果顶点v是一个叶子结点，那么将叶子结点加入独立集，之后删除叶子节点与其父亲相关的所有边。

如果是带权的独立集问题，也就是node有权重，贪心算法将失效，但是可以使用一个动态规划的算法解决问题。

Dynamic programming solution. Root tree at some node, say r.

- $OPT_{in}(u)$ = max weight independent set of subtree rooted at u, containing u.
- $OPT_{out}(u)$ = max weight independent set of subtree rooted at u, not containing u.

$$OPT_{in}(u) = w_u + \sum_{v \in \text{children}(u)} OPT_{out}(v)$$

$$OPT_{out}(u) = \sum_{v \in \text{children}(u)} \max\{OPT_{in}(v), OPT_{out}(v)\}$$

```
Weighted-Independent-Set-In-A-Tree(T):
   Root the tree at a node $r$
   for each node $u$ of $T$ in postorder
      if $u$ is a leaf
         $M_{in}[u] \leftarrow w_u$
         $M_{out}[u] \leftarrow 0$
      else
         $M_{in}[u] \leftarrow \sum_{v \in \text{children}(u)} M_{out}[v] + w_u$
         $M_{out}[u] \leftarrow \sum_{v \in \text{chilren}(u)} \max(M_{out}[v], M_{in}[v])$
   return $\max(M_{in}[r], M_{out}[r])$
```

ensures a node is visited after all its children

Circular Arc Covering

Given a graph $G = (V, E)$, and m paths $p\#, p7, \ldots, p8$, assign each path a color so that any two paths that share an edge must have different colors. The goal is to use as few colors as possible.

Runningtime. $O(k!\cdot n)$

- n phases of the algorithm.
- Bottleneck in each phase is enumerating all consistent colorings.

•There are at most k intervals through $v*$, so there are at most $k!$ colorings to consider. **Remark.** This is poly(n) time if k=O(log$n$/loglog$n$)

•This algorithm is practical for small values of k (say k = 10) even if the number of nodes n (or paths) is large

Gradient descent

Vertex cover. Given a graph G = (V, E), find a subset of nodes S of minimal cardinality such that for each (u, v) ∈ E, either u or v (or both) are in S

Gradient descent. Start with S = V. If there is a neighbor S′ that is a vertex cover and has lower cardinality, replace S with S′

Maximum Cut

最大割问题是在图中找一个最大的割，最大割是全局找最大，是一个NP难的问题，使用局部搜索的方法近似解决。首先是single–flip neighborhood算法，首先随机产生一个割之后循环，如果移动一个node到另一边能够提高割，就移动。

Let (A, B) be a locally optimal partition and let (A*, B*) be the optimal partition.

Then w(A, B) ≥ ½ Σ_e w_e ≥ ½ w(A*, B*).

Weights are nonnegative

Pf.

- Local optimality implies that for any u ∈ A : $\sum_{v\in A} w_{uv} \le \sum_{v\in B} w_{uv}$
  Adding up all these inequalities yields:

$$2 \sum_{\{u,v\}\subseteq A} w_{uv} \le \sum_{u\in A,\ v\in B} w_{uv} = w(A,B)$$

- Similarly

$$2 \sum_{\{u,v\}\subseteq B} w_{uv} \le \sum_{u\in A,\ v\in B} w_{uv} = w(A,B)$$

- Now,

each edge counted once

$$\sum_{e\in E} w_e = \underbrace{\sum_{\{u,v\}\subseteq A} w_{uv}}_{\le\frac{1}{2}w(A,B)} + \underbrace{\sum_{u\in A,\, v\in B} w_{uv}}_{w(A,B)} + \underbrace{\sum_{\{u,v\}\subseteq B} w_{uv}}_{\le\frac{1}{2}w(A,B)} \le 2w(A,B)$$

▪

Big-improvement-flip algorithm Only choose a node which, when flipped, increases the cut value by at least (2e/n)*W(A,B)

Upon termination, big-improvement-flip algorithm returns a cut (A, B) with (2+e) w(A,B)3 w(A*,B*).

Let (A, B) be a locally optimal partition and let (A*, B*) be the optimal partition.

partition. Then $w(A, B) \geq \frac{1}{2} \Sigma_e w_e \geq \frac{1}{2} w(A^*, B^*)$

Big-improvement-flip algorithm. Only choose a node which, when flipped, increases the cut value by at least $\frac{2\varepsilon}{n} w(A,B)$

Claim. Big-improvement-flip algorithm terminates after $O(\varepsilon^{-1} n \log W)$ flips where $W = \Sigma_e w_e$.

- Each flip improves cut value by at least a factor of $(1 + \varepsilon/n)$
- After n/ε iterations the cut value improves by a factor of 2
- Cut value can be doubled at most $\log_2 W$ times.

Observation. In general, there can be many Nash equilibria. Even when it's unique, it does not necessarily equal the social optimum.

Price of stability. Ratio of best Nash equilibrium to social optimum. 最佳纳什均衡与社会最优的比率。

Price of stability = Q(log k).

•**Social optimum:** Everyone takes bottom paths. **Unique Nash equilibrium:** Everyone takes top paths. **Price of stability:** H(k) / (1 + e).

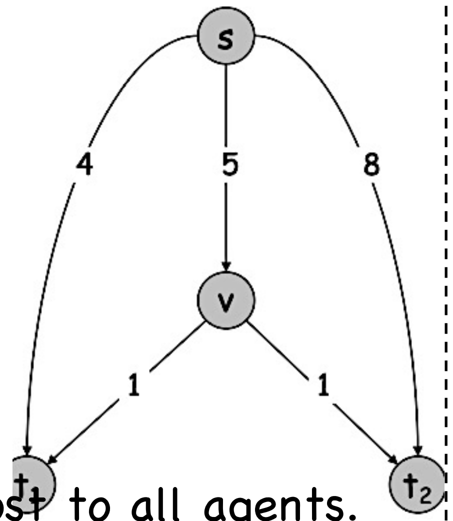
$H(k)=\sum_{i=1}^{k} \frac{1}{i}$

Pf. Consider a set of paths $P_1, \ldots, P_k$.

- Let x_e denote the number of paths that use edge e.
- Let $\Phi(P_1, \ldots, P_k) = \Sigma_{e \in E}\, c_e \cdot H(x_e)$ be a potential function.
- Since there are only finitely many sets of paths, it suffices to show that Φ strictly decreases in each step.

Nash Equilibria 纳什均衡

多播问题中，从s原点出发到多个代理，如果一条路被k个代理使用这这条路的负担将被除以k，这导致代理选择路线变化。如右图，设一开始都从外部出发，这时候t2将使用v中间，变为6，这时候t1分析发现如果公用5会变成2.5，也开始转化，最后两个都用中间，达到纳什均衡



Social optimum. Minimizes total cost to all agents.

Consider agent j switching from path P_j to path P_j'.
Agent j switches because

$$\underbrace{\sum_{f \in P_j' - P_j} \frac{c_f}{x_f + 1}}_{\text{newly incurred cost}} < \underbrace{\sum_{e \in P_j - P_j'} \frac{c_e}{x_e}}_{\text{cost saved}}$$

Φ increases by

$$\sum_{f \in P_j' - P_j} c_f \left[H(x_f + 1) - H(x_f) \right] = \sum_{f \in P_j' - P_j} \frac{c_f}{x_f + 1}$$

Φ decreases by

$$\sum_{e \in P_j - P_j'} c_e \left[H(x_e) - H(x_e - 1) \right] = \sum_{e \in P_j - P_j'} \frac{c_e}{x_e}$$

Thus, net change in Φ is negative. ▪

Amortized Analysis 平摊分析

平摊分析有三种方法：聚类分析 Aggregate analysis，记账方法 Accounting method，势能方法 Potential method。聚类分析：证明对所有的n，由n个操作所构成的序列的总时间在最坏情况下为T(n)，每一个操作的平均成本为T(n)/n；记账方法：在平摊分析的记帐方法中，决定每一个操作的均摊成本credit，对不同的操作赋予不同的费用，某些操作的费用比它们的实际代价或多或少:。势能方法：在平摊分析中，势能方法而是将存款总体上表示成一种“势能”或“势”比如，动态表分析。

聚类分析

考虑MULTIPOP操作的栈，PUSH 和 POP都是O1的时间，但一连串的n个PUSH和POP的总消耗为n，这时候我们分析，我们发现POP的1次数一定小于或者等于PUSH的次数，Tn=sum(Ci)=#Push+#POP<=2*#PUSH<=2n 那么根据聚类分析1/nTn为成本，那么实际上MULTI是O1。分析二进制计数器，使用数组记录，粗略分析一个增加操作可能改变所有的k位，那么Tn<=kn，聚类分析基础操作有flip10和flip01，在n次中数组A0位置每一次操作都会flip因此n次，而A1是每两次flip也就是n/2，Ai flip n/2^i向下取整次，所以求和为n+n/2+n/4<=2n还是O1 从例子中我们可以得知，需要分析出基础的步骤类，之后求和计算。

记账方法

每次一个均摊成本Ci’，如果Tn<=sum(Ci’)如果Ci’>Ci那么额外的部分可以存储下来为存款credit，如在MULTIPOP中，我们设PUSH花费2，其他为0，那么最多花费2nPUSH。在二进制里可以设flip01为2费，10为0费所以还是小于2n。作业中，双栈成队问题一个元素最多被push2次pop2次共为4，分配到insert上可以为3push2次pop1次，而1次作为删除

势能方法

势能方法为当前状态的势能，那么均摊分析Ci’=Ci + phi(Si)-phi(Si-1)那么sum(Ci’)=sum(Ci)+phi(Sn)-phi(S0)比如在MULTIPOP中，设置phi为数组中的元素个数，那么有PUSH=Ci+phi(i)-phi(i-1)=1+1=2类似的有POP为0 MPOP为多个POP = k*0 = 0所以最多还是2n，在二进制中，设1的个数为势能函数，那么从0开始，n个increase和为2n。

动态表

比如我们设置势能函数为,设num为当前数组内元素个数，size为整个数组的长度，设置势能函数phi=2*T.num-T.size 分两种情况讨论，如果i操作没有触发操作，那么size(i-1)=size(i) num(i)-1=num(i-1)那么均摊成本有Ci’=Ci+phi(i)-phi(i-1)=1+(2numi-sizei)-(2numi-1-sizei-1)=3,类似的如果出发了，就修改num和size，得到结果。作业中的备份数组问题每n次备份一次数组，设置phi=i mod n之后分情况讨论

随机算法

Las Vegas总生成正确答案，但运行时间可能是随机的，Monte Carlo算法时间是相同的但是结果可能是错误的。
如果反复进行独立实验，每次实验成功的概率p>0那么直到第一次成功期望的实验次数为1/p

Max Cut

Max Cut有一简单的2approximation Monte Carlo算法将一个node放到随机的一边正确的概率为1/2

$P[e \text{ in } cut]=P[(i \text{ in } L) \wedge (j \text{ in } R)]+Pr[(j \text{ in } L) \wedge (i \text{ in } R)]$
$=1/4+1/4=1/2$. $E[X_e]=1/2$ $E[X]=e/2$

Contention Resolution 消除竞争

假设有n个进程竞争访问一个数据库，采用一种方法，在每一轮次中，每个进程有p概率访问数据库成功为p(1-p)^(n-1)如果p设置为1/n则数学上成功概率Pr收敛到1/e渐进的等于\Theta(1/n),针对某一进程，如果在t个时间段都失败了，那么就是(1-1/en)^t概率，如果设置t=en那么根据数学收敛到1/e表明如果en轮中都失败的概率以1/e为上界，如果小幅度提高t=(en)(cln n)得到概率n^(-c)，这时候我们考虑至少经过多少轮能够让所有进程都反问到，假设简单独立分布，就是概率的和那么这时候如果特别的t=2(en)(cln n)有P为sum<n*n^-2=1/n都失败，结论就是如果这个t，所有进程至少成功访问一次数据库的概率至少为1-1/n

$1/4<(1-1/n)^n<1/e<(1-1/n)^{n-1} <1/2$

$9/4 < (1 + 1/n)^n < e < (1 + 1/n)^{n+1} < 27/8$

全局最小割

全局最小不同于定起点终点最小，而是找所有割中最小的割，使用收缩算法，每次任意选择一条边进行收缩，保留平行边，收缩直接连接边，删除自循环，最后返回两个超级点之间的割，算法返回正确答案的概率大于2/n方，假设最小割为k。那么显然每一个节点的度数最小为k，一个度有两个边，那么|E|>=kn/2，所以收缩一条边的概率为k/(kn/2)=2/n，现在考虑j次迭代，当前存在n-j个超级节点，那么至少有k(n-j)/2条边，那么下一轮迭代，收缩一条边的概率最多为k/(k(n-j)/2)=2/(n-j)，那么连乘非事件就是下届，计算出来为2/(n(n-1))那么之后通过重复运行算法(1-1/x)^x小于1/e，重复n方次，很容易将算法错误率下降到1/e下

猜牌

无记忆猜牌每次成功的概率都是1/n那么猜对的期望只有1(n*1/n)如果是有记忆猜牌，猜对的概率逐步上升为1/j就是调和级数求和，接近ln(n+1)那么猜对的期望升高，是\Thetalogn的次数水平

生日悖论 The Brithday Paradox

对比在一堆人找和一个确定人的生日相同的概率很低，在多个人里面找一对生日相同的概率要高很多，考虑找不到计算都不同的概率，那么概率为1*(1-1/365)*(1-2/365)...(1-(n-1)/365) 1减去不同概率为存在概率，之后期望为k(k-1)/2n>1 k是至少多少。

优惠卷收集 Coupon Collector

每盒一张优惠券有 n 种不同类型的优惠券，需要打开多少个盒子才能有每种类型的优惠券。开箱遇到没有的概率会越来越小，所以取得进展的概率为(n-j)/n每个步骤j的期望是n/(n-j)那么总体上的期望有n*sum(1/(n-j))=\Theta(nlogn)

MAX 3-SAT

在MAX 3–SAT中希望知道可以满足多少SAT子句，希望找到一个赋值最多满足，这是一个NP难问题而一个子句为假的概率为1–(1/2)^3=7/8那么和起来为7/8k个子句也就是随机选也有7/8

Quicksort 快排 Rquicksort 随机快速排序

随机排序下，一个随机算法随机分配为1/4和3/4则有Tn=T(1/4n)+T(3/4n)+n那么计算出Tn=\Theta(nlogn)

Hash Tables

Closed addressing方法在插入时加入一个链表中Load factor为n/m平均一个位置有n/m个key，选择哈希函数是非常重要的，比如常见的mod函数，常选择m为质数靠近2的幂Division method另一种是Multiplication method是下取整(m(kAmod1))A是一个常数，knuth建议选择根号5–1/2 Universal hashing用散列使用随机散列函数，而不是使用固定的散列函数，该函数是从某组family函数H中选择的。如果Hxi=Hx=1/m那么Ex=n/m，第一种方法选择一个质数p且p>m和p>all keys其中一个函数为((ak+b)modp)modm,H函数族为a取从1到p–1和b从0到p–1

Proof Let $x, y < p$ be two different keys. For a given h_{ab} let

$$r = (ax + b) \bmod p, \qquad s = (ay + b) \bmod p$$

We have $r \neq s$, because $r - s \equiv a(x - y) \bmod p \neq 0$, since neither a nor $x - y$ divide p.

Also, each pair (a, b) leads to a different pair (r, s), since

$$a = \left((r - s)(x - y)^{-1} \bmod p\right), \qquad b = (r - ax) \bmod p$$

- Here, $(x - y)^{-1} \bmod p$ is the unique multiplicative inverse of $x - y$ in $\mathbb{Z}_p^*$.

Hash family 2

Since there are $p(p-1)$ pairs (a,b) and $p(p-1)$ pairs (r,s) with $r \neq s$, then a random (a, b) produces a random (r, s).

The probability x and y collide equals the probability $r \equiv s$ mod m. For fixed r, number of $s \neq r$ s.t. $r \equiv s$ mod m is $(p-1)/m$.

So for each r and random $s \neq r$, probability that $r \equiv s \bmod m$ is $((p-1)/m))/(p-1) = 1/m$.

So $\Pr_{h_{ab} \in H_{pm}}[h_{ab}(x) = h_{ab}(y)] = 1/m$ and H_{pm} is universal.

Perfect hashing uses two levels of universal hashing.

- The first layer hash table has size m = n.
- Use first layer hash function h to hash key to a location in T.
- Each location j in T points to a hash table S_j with hash function h_j.
- If n_j keys hash to location j, the size of S_j is $m_j = n_j^2$.

We'll ensure there are no collisions in the secondary hash tables $S_1, \ldots, S_m$.

- So all operations take worst case O(1) time.

Overall the space use is $O(m + \sum_{j=1}^{m} n_j^2)$.

- We'll show this is $O(n) = O(m)$.
- So perfect hashing uses same amount of space as normal hashing.

Approximation Algorithms – Set Covering

在这里每个集合都有权重这里追求最小权重，那么这里每次选择一个集合U，能够让当前的wi/|S交U|

Approximation Algorithms – Scheduling

如果S的某个子集与t求和，则最小完工时间为S–t可以设计一个二近似的算法，随机任意排列任务，之后分配给当前没有任务的机器。首先最佳完工时间最少是时间之和/机器数量，一个更实用的是最佳时间必然>=任务的最大时间，而当前总负载Ti–tj<=1/m sum(Tk) 也就是<T*最优，另一个界是tj一定小于T*加起来也就是Ti一定小于2T*，可以使用LPT也就是最长优先来进一步缩小近似比到3/2，证明为如果任务比m个作业少，贪心会得到最优，如果大于m，那么又一个下界为T*>=2tm+1（最大最先，其中必然一个机器分配到两个任务因为任务比m大），假设机器至少有两个作业，那么tj是最后一个作业，那么由于最大最先所以tj<=tm+1<=1/2T*，有tj<=T*所以得到3/2

Claim 1 LPT's makespan = T+t ≤ M*+t.

- As in LS, no processor is idle up to time T, so M* ≥ T.

Case 1 t ≤ M*/3.

- Then LPT's makespan ≤ M* + t ≤ M* + M*/3 = 4/3 M*.

Case 2 t > M*/3.

- Since X is the smallest task, all tasks have size > M*/3.
- So the optimal schedule has at most 2 tasks per processor. So n ≤ 2m.
- If 1 ≤ n ≤ m, then LPT and optimal schedule both put one task per process
- If m < n ≤ 2m, then optimal schedule is to put tasks in nonincreasing order 1,…,m, then on m,…,1.
  - LPT also schedules tasks this way, so it's optimal.

Approximation Algorithms – 背包问题

使用动态规划算法，时间复杂度是伪多项式O(V*n^2) V*有可能很大，这个算法就不是多项式的，这时候就可以缩小大的V就可以加快速度，对于任何e大于0可以找到一个1+e的近似算法，时间变为O(n^3/e)让变化比例为eV*/2n其中V*是最大的V，之后scale所有的value为v/比例向上取整。

Let S* be the optimal solution to the original problem.

$$\sum_{i\in S^*} v_i \le \sum_{i\in S^*} u_i \qquad u_i \ge v_i$$

$$\le \sum_{i\in S} u_i \qquad \text{S is optim}$$

$$\le \sum_{i\in S} (v_i + \theta) \qquad u_i \le v_i+\theta$$

$$\le \sum_{i\in S} v_i + n\theta \qquad |S| \le n$$

$$n\theta = \frac{\varepsilon}{2} v_j \le \frac{\varepsilon}{2} u_j \le \frac{\varepsilon}{2} \sum_{i\in S} u_i$$

$$\sum_{i\in S} v_i \ge \sum_{i\in S} u_i - n\theta \ge \left(\frac{2}{\varepsilon} - 1\right) n\theta$$

$$\sum_{i\in S^*} v_i \le \sum_{i\in S} v_i + n\theta \le \sum_{i\in S} v_i + \varepsilon \sum_{i\in S} v_i = (1+\varepsilon)\sum_{i\in S} v_i$$

$$n\theta \le \varepsilon \sum_{i\in S} v_i$$

Approx Algorithms – vertex cover

随机算法，每次选择一个随机的边，之后这条边的端点加入结果，之后删除所有uv相关边，这是一个2近似，

None of the edges in A touch each other.Each time we pick an edge, we remove all adjacent edge.So vertex in C* covers at most one edge in A. edges covered by a vertex all touch each other. Every edge in A is covered by a vertex in C*. Because C* is a vertex cover. So |C*| ≥ |A|. the algorithm uses is 2|A|. So (#algorithm uses) / (opt cover) = 2|A| / |C*| ≤ 2|A| / |A| = 2.

Approximation Algorithms – TSP

TSP是NP难的，但是存在一个简单的2近似，还有一个1.5近似2近似算法中，首先建立一颗MST最小生成树，之后使用深度优先算法便利这颗树，之后按照深度优先遍历的算法的顺序产生一个不重复的任意一个顶点之后返回H为一个TSP tour，让H*为一个最优TSP，那么如果我们删除一条边，就变成了一颗生成树，我们的算法是MST所以H*树必<=MST,由由于是深度优先，遍历一个边两次，所以c(H) ≤ c(T′) = 2 c(T) ≤ 2 c(H*). 所以是2近似。一种更优秀的算法是使用欧拉图，首先生成MST之后寻找奇数度的顶点T，之后构建最小cost perfect matching M在V‘上，之后寻找一个欧拉路，之后建立汉密尔顿回路作为TSP

Chernoff Bounds

Markov’s Inequality Given a positive random variable X, Pr $[X \geq a] \leq E[X]/a$ for any $a > 0$.

Chebychev’s Inequality Given a random variable X and any $a > 0$, we have Pr$[|X - E[X] \geq a|] \leq Var[X]/a$^2.

$$\text{For } 0 < \delta \leq 1,\ \mathbf{Pr}[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}.$$

$$\text{For } \delta > 1,\ \mathbf{Pr}[X \geq (1+\delta)\mu] \leq e^{-\mu\delta\ln\delta/3}.$$

$$\text{For } 0 \leq \delta \leq 1,\ \mathbf{Pr}[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}.$$

Let X, $X_1,\ldots,X_n$ be defined as earlier. Then for any $\delta>0$

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

Let $X_1,\ldots,X_n$ be independent {-1,1} valued random variables, with $\Pr[X_i=1]= \Pr[X_i=-1]=1/2$

$X = \sum_i X_i$. Then for any $\delta \geq 0$, $\Pr[X \geq \delta] = \Pr[X \leq -\delta] \leq e^{-\frac{\delta^2}{2n}}$.

Load Balancing 负载均衡

随机算法简单，任意一个job分配给一个随机的计算机，那么job的概率为1/n，一个计算机i得到的任务期望为求和Yij，可以用bound分析

First consider the case $m = n$.

We have $\mu = \mathbf{E}(X_i) = \mathbf{E}[\sum_{j=1}^{m} Y_{ij}] = \sum_{j=1}^{m} \mathbf{E}[Y_{ij}] = m\frac{1}{n} = 1$ for every i.

We’ll show $\mathbf{Pr}[X > O(\frac{\ln n}{\ln\ln n})] < \frac{1}{n}$.

- So with high probability ($> 1 - 1/n$), every computer gets at most $O(\frac{\ln n}{\ln\ln n})$ times its expected number (=1) of jobs.

Let’s focus on X_i for some particular i. Let $\delta = O\left(\frac{\ln n}{\ln\ln n}\right) > 1$.

By part 2 of Theorem 1, $\mathbf{Pr}[X_i > (1+\delta)\mu] \leq e^{-\delta\ln\delta/3}$, since $\mu = 1$.

- So $e^{-\delta\ln\delta/3} = e^{O(\ln n)} \leq \frac{1}{n^2}$ for some choice of the constant in the big-O.

We have $\mathbf{Pr}[X_i > O(\frac{\ln n}{\ln\ln n})] \leq \frac{1}{n^2}$ for every i. Hence by the union bound $\mathbf{Pr}[X_i > O(\frac{\ln n}{\ln\ln n})$ for any $i] \leq n\frac{1}{n^2} = \frac{1}{n}$.

So $\mathbf{Pr}[X = \max_i X_i > O(\frac{\ln n}{\ln\ln n})] < \frac{1}{n}$.

We see that when there are n jobs for n computers, the load can be quite unbalanced.

Suppose now we have more jobs, $m = 16n\ln n$. Then $\mu = 16\ln n$.

By Theorem 2, $\mathbf{Pr}[X_i > 2\mu] < \left(\frac{e}{4}\right)^{16\ln n} < \left(\frac{1}{e^2}\right)^{\ln n} = \frac{1}{n^2}$.

Thus, by the union bound $\mathbf{Pr}[\max_i X_i > 2\mu] \le n\frac{1}{n^2} = \frac{1}{n}$.

By part 3 of Theorem 1, $\mathbf{Pr}[X_i < \frac{1}{2}\mu] \le e^{-(\frac{1}{2})^2 16\ln n/2} = e^{-2\ln n} = \frac{1}{n^2}$.

So by the union bound, $\mathbf{Pr}[\min_i X_i < \frac{1}{2}\mu] \le \frac{1}{n}$.

Thus, we have that with high probability, every computer has between half and twice the average load.

These results hold for any $m > 16n\ln n$. So the more jobs there are, the better the load balancing random allocation achieves. This is similar to the phenomenon where the more coins we flip, the more likely we are to get the expected number of heads (in relative terms).

Consider an item i. Suppose i belongs to k different sets.

For the j'th such set S, define X_j=1 if S is in the first group, and X_j=-1 if it's in the other group.

The imbalance from item i is simply $B_i = \sum_{j=1}^{k} X_j$.

Since sets are assigned randomly, $X_1,\ldots,X_k$ are independent {1,-1} valued random values, we can apply Thm 3 to bound $\max_i B_i$.

If $k \le \sqrt{4m\ln n}$, then $B_i \le \sqrt{4m\ln n}$

If $k > \sqrt{4m\ln n}$, then by Theorem 3 we have

$$\begin{aligned}\Pr[B_i > \sqrt{4m\ln n}] &\le e^{-(\sqrt{4m\ln n})^2/(2k)} \\ &= e^{-4m\ln n/(2k)} \\ &\le e^{-4m\ln\frac{n}{2m}} \\ &= e^{-2\ln n} \\ &= \frac{1}{n^2}\end{aligned}$$

We showed the probability item i's balance is $\ge \sqrt{4m\ln n}$ is $\le \frac{1}{n^2}$. By the union bound, the probability that any of the n items' imbalance is $\ge \sqrt{4m\ln n}$ is $\frac{1}{n}$. So with high probability the max imbalance is $\le \sqrt{4m\ln n}$

Set Balancing

有一个自然数集合1到n，还有m个集合都是这个子集合，可以使用二进制表示如{1,3,4}为[1,0,1,1]我们希望将集合分成两组，使得组中集合的总和大致相等，找到一个尽量好的，随机算法是随机分配到不同的group高概率最大imbalance为根号(4mlnn)

Bloom Filter

设一开始一个全0数组，如果有一个数字，使用k个独立的哈希函数哈希，将k个哈希值作为下标，标注数组全部为1，查询的时候，同样计算哈希值，如果有一个0那么其必不存在，如果全为1，那么有很大可能数字存在。假阳性率与k，m表size，n key个数相关，任何一个位置为1的概率为1/m，那么为0为1–1/m那么k哈希之后0 $(1 - 1/m)$^k 为n个key就是(1–1/m)^nk接近于e的–nk/m次方,变为1就是1减这个值

m positions in the array, assume (1-p)m positions are 1.

argument rigorous by showing that the actual number 1's $(1-p)m \pm \sqrt{m\log m}$ with high probability.

假阳性when we check k random locations, they're all 1

Probability is $f = (1-p)^k \approx \left(1 - e^{-\frac{nk}{m}}\right)^k$.

The optimum k is $\frac{m\ln(2)}{n}$, which leads to $f = \left(\frac{1}{2}\right)^k \approx 0.6185^{\frac{m}{n}}$. 目前bloom无法处理删除

Fingerprinting 数据库

将A与B的数据库看作二进制，之后看为二进制数组序列(x1..xn)之后计算x=sum(xi*2^(i−1))之后找一个质数p计算xmodp作为指纹，这样A和B就可以进行比对，传输只需要O log p bits。任何t最多有log2(t) 个不同的主除数。a−b has most n distinct prime divisors.

作业概述

MAX覆盖问题，选择k个子集合使其能够覆盖最多元素，使用贪心算法，每次选择一个集合中最多没有被覆盖的元素.

cover. The set we add in iteration i + 1 can only cover more new elements than this set, i.e. we add a set covering at least n_i/k elements. So we have $n_i{+}1 \le n_i{-}n_i/k = n_i(1{-}1/k)$. In turn, we have $n_k \le (1{-}1/k)^k OPT \le \frac{1}{e}OPT$, or equivalently our solution covers at least (1-1/e)OPT elements.

K Centers 算法

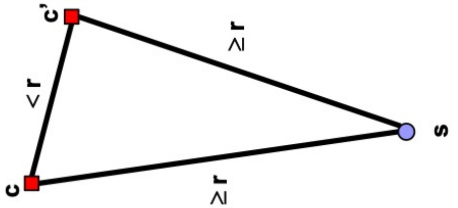
有一个简单的2近似算法，从空集合开始，每次选择一个site距离当前集合距离最大，之后将这个site作为新中心加入到答案C

Let C be the algorithm's output, and r be C's radius.

- $r = max_{s\in S} min_{c\in C} d(s,c)$

Lemma 1 For any $c, c' \in C, d(c,c') \ge r$.

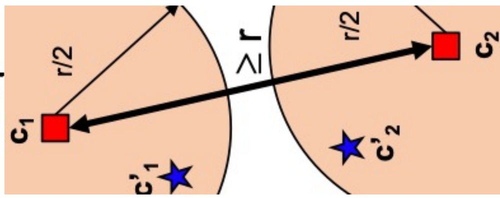
Proof Since r is the radius, there exists a point $s \in S$ at distance ≥ r from all the centers.



- If there's no such s, then C's radius < r.
- So s is distance ≥ r from c and c'.
- Suppose WLOG c' is added to C after c.
- If d(c,c')<r, then algorithm would add s to C instead of c', since s is farther.

Cor There exist k+1 points mutually at distance ≥ r from each other.

- By the lemma, the k centers are mutually ≥ r distance apart.
- Also, there's an s∈S at distance ≥ r from all the centers.
  - Otherwise, C's covering radius is < r.
- So, the k centers plus s are the k+1 point



Call these k+1 points D.

Let C* be an optimal cover with radius r*.

Lemma 2 Suppose r > 2r*. Then for every c∈D, there exists a corresponding c'∈C*. Furthermore, all these c' are unique.

Proof Draw a circle of radius r/2 around each c ∈ D.

- There must be a c' ∈ C* inside the circle, because
  - c is at most distance r* away from its nearest center, since r* is C*'s radius.
  - r/2>r*.
- Given $c_1,c_2 \in D$, let $c'_1,c'_2 \in C^*$ be inside c_1 and c_2's circle, resp.
- c_1 and c_2's circles don't touch, because $d(c_1,c_2) \ge r$.
- So $c'_1 \ne c'_2$

Thm Let C be the output of Gonzalez's algorithm and let C* be an optimal k-center. r(C) ≤ 2r(C*).

Proof By Lemma 2, if r(C)>2r(C*), then for every c∈D, there is a unique c'∈C*.

- But there are k+1 points in D, by the corollary.
- So, there are k+1 points in C*. This is a contradiction because C* is a k-center.

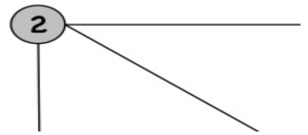
FPT 习题概述

汉密尔顿路径问题实际上可以在O(2^n*n^3)内解决

Weighted vertex cover. Given a graph G with vertex weights, find a vertex cover of minimum weight.
It's a special case of the set cover problem, so the H(d*) approximation ratio can be achieved by the greedy algorithm, where d* = max degree

Pricing method. Each edge must be covered by some vertex i. Edge e pays price p_e 3 0 to use vertex i.

Edges incident to vertex i should pay $\le w_i$ in total.



$$\textit{for each vertex } i\text{:}\quad \sum_{e=(i,j)} p_e \le w_i$$

For any vertex cover S and any fair prices p_e: $\sum_e p_e \le w(S)$.

$$\sum_{e\in E} p_e \le \sum_{i\in S} \sum_{e=(i,j)} p_e \le \sum_{i\in S} w_i = w(S)$$

each edge e covered by at least one node in S

sum fairness inequalities for each node in S

```
Weighted-Vertex-Cover-Approx(G, w) {
   foreach e in E
      pₑ = 0

   while (∃ edge i-j such that neither i nor j are tight)
      select such an edge e
      increase pₑ without violating fairness
   }

   S ← set of all tight nodes
   return S
```

$$\sum_{e=(i,j)} p_e = w_i$$

Pricing method is a 2-approximation.

Algorithm terminates since at least one new node becomes tight after each iteration of while loop.

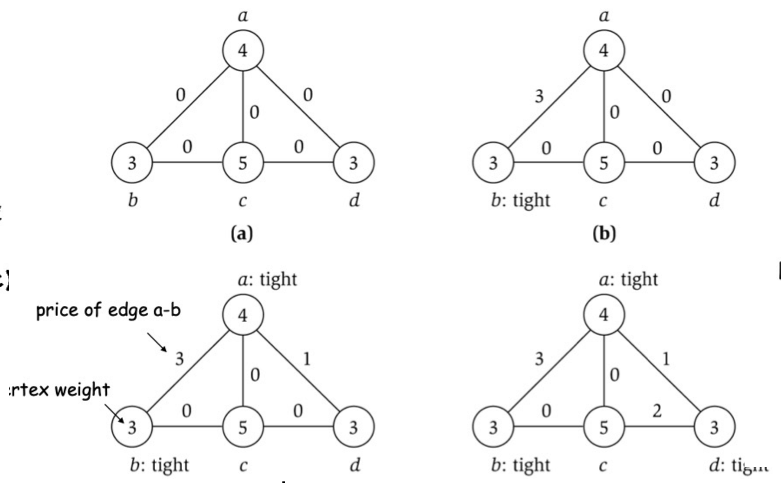
Let S = set of all tight nodes upon termination of algorithm. S is a vertex cover: if some edge i-j is uncovered, then neither i nor j is tight. But then while loop would not terminate.

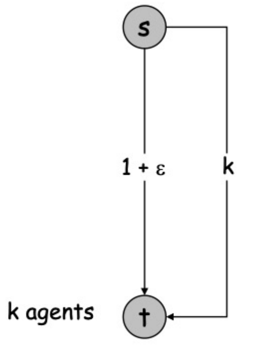
S* be optimal vertex cover. We show w(S) <= 2w(S*).

$$w(S) = \sum_{i\in S} w_i = \sum_{i\in S} \sum_{e=(i,j)} p_e \le \sum_{i\in V} \sum_{e=(i,j)} p_e = 2\sum_{e\in E} p_e \le 2w(S^*).$$

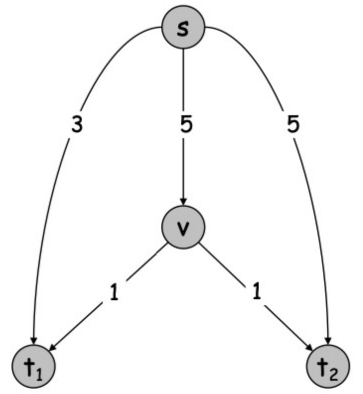
all nodes in S are tight | $S \subseteq V$, prices ≥ 0 | each edge counted twice | fairness lemma

Social optimum. Minimizes total cost to all agents.





Social optimum = 1 + ε
Nash equilibrium A = 1 + ε
Nash equilibrium B = k



Social optimum = 7
Unique Nash equilibrium = 8

NAE3SAT，是指一个满足条件的 assignment 不仅要使得表达式为真，还需要对于SAT表达式中的每个分句，为真的变量不超过2个

从3SAT问题到n 3SAT的映射。将3SAT的每一个分句(y1∨y2∨y3)，转化为nae 3SAT的两个分句(y1∨y2∨zi)∧(¯zi∨y3∨b)。其中zi是对应原来每个分句都有一个新变量，而b则是全局变量。

证明，若3SAT为真则nae 3SAT为真。若3SAT为真，要么y1,y2中一个为真，此时令zi=0，若y3为真，则令zi=1，令b=0。如此既保证了每个nae 3SAT的分句为真，也保证了没有一个分句的三个变量全为真n3SAT为真则3SAT为真n3SAT为真，对应 assignment 中若b=1，则取该 assigment 的否定（显然对于nae 3SAT，解assignment 的否定也是一个解）。该解中每两个分句，选取 z 变量为0的一个分句，其中必存在一个 y=1，满足原 SAT