LP rounding technique:
Write weighted v.c. problem as integer linear program. Min E x_u w_u (x_u is variable and belongs to 0,1). w_u given.
For each edge x_u + x_v >= 1
Relax integrality constraints to get an LP.
Replace x_u {0,1} with x_u >= 0 for all u.
Round fractional solution to get a solution in your integral feasible space. If x_u >= .5, add to vertex cover. x < .5, don't add to vc.
Ovjective goes up by at most 2x because you replaced all x>= .5 by 1 and just got rid of all x_u < .5

Randomized Algorithm:
E[x] = expectation of x
E Pr(e) X (e)
Linearity of expectation:
E[x+y] = E[x] + E[y]
X_i = 0 if event doesn't happen, =1 if it does happen. E[x] = 0*Pr(x = 0) + 1*Pr(x = 1) = Pr(x=1)
Markows:
P(x >= t) <= E[x]/t
Var(x) = E[x^2] – (E[x])^2
Chebyhev:
Pr[|X-E[x]| > t] < var{x}/t^2

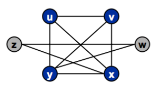
**Lemma 2.1.5** $\mathbf{E}[\textit{num. edges in cut}] = \frac{m}{2}$

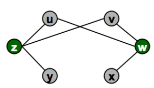
**Proof:** Let us number the edges 1 to $m$. Define an indicator variable $X_i$ for each edge $i$ s.t. $X_i = 1$ if the edge crosses the cut and $X_i = 0$ if the edge doesn't cross the cut. Since we assigned the vertices independently randomly, probability that both endpoints of $i$ are in the same set = probability that the endpoints are in different sets = $\frac{1}{2}$. Hence, $\mathbf{E}[X_i] = \frac{1}{2}$. Expected total number of edges crossing the cut = $\sum_{i=1}^{m} \mathbf{E}[X_i] = \frac{m}{2}$ by linearity of expectation. ■ Hence the random algorithm is a 2-approx algorithm on expectation.

## Vertex Cover and Clique

Claim. VERTEX COVER $\equiv_P$ CLIQUE.
- Given an undirected graph G = (V, E), its complement is G' = (V, E'), where E' = { (v, w) : (v, w) ∉ E}.
- G has a clique of size k if and only if G' has a vertex cover of size |V| - k.

Proof. ⇒
- Suppose G has a clique S with |S| = k.
- Consider S' = V – S.
- |S'| = |V| - k.
- To show S' is a cover, consider any edge (v, w) ∈ E'.
  - then (v, w) ∉ E
  - at least one of v or w is not in S (since S forms a clique)
  - at least one of v or w is in S'
  - hence (v, w) is covered by S'



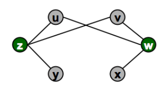



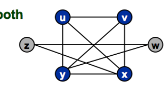

11

## Vertex Cover and Clique

Claim. VERTEX COVER $\equiv_P$ CLIQUE.
- Given an undirected graph G = (V, E), its complement is G' = (V, E'), where E' = { (v, w) : (v, w) ∉ E}.
- G has a clique of size k if and only if G' has a vertex cover of size |V| - k.

Proof. ⇐
- Suppose G' has a cover S' with |S'| = |V| - k.
- Consider S = V – S'.
- Clearly |S| = k.
- To show S is a clique, consider some edge (v, w) ∈ E'.
  - if (v, w) ∈ E', then either v ∈ S', w ∈ S', or both
  - by contrapositive, if v ∉ S' and w ∉ S', then (v, w) ∈ E
  - thus S is a clique in G



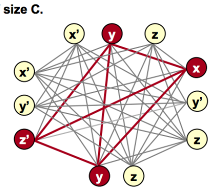




12

## Satisfiability Reduces to Clique

Claim. CNF-SAT $\le_P$ CLIQUE.
- Given instance of CNF-SAT, create a person for each literal in each clause.
- Two people know each other except if:
  - they come from the same clause
  - they represent a literal and its negation
- Clique of size C ⇒ satisfiable assignment.
- Satisfiable assignment ⇒ clique of size C.
  - (x, y, z) = (true, true, false)
  - choose one true literal from each clause

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')
C = 4 clauses

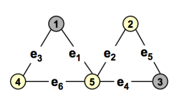



## Subset Sum

Claim. G has vertex cover of size k if and only if there is a subset S that sums to exactly t.

Proof. ⇒
- Suppose G has a vertex cover C of size k.
- Let S = C ∪ { $y_j$ : $|e_j \cap C| = 1$ }
  - most significant bits add up to k
  - remaining bits add up to 2

| | | $e_1$ | $e_2$ | $e_3$ | $e_4$ | $e_5$ | $e_6$ | decimal |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 5,184 |
| $x_2$ | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 4,356 |
| $x_3$ | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 4,116 |
| $x_4$ | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 4,161 |
| $x_5$ | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 5,393 |
| $y_1$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1,024 |
| $y_2$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 256 |
| $y_3$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 64 |
| $y_4$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 16 |
| $y_5$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 4 |
| $y_6$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| t | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 15,018 |
| | k | | | | | | | |



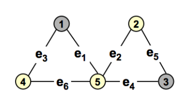

67

## Subset Sum

Claim. G has vertex cover of size k if and only if there is a subset S that sums to exactly t.

Proof. ⇐
- Suppose subset S sums to t.
- Let C = S ∩ {$x_1$, . . . , $x_n$}.
  - each edge has three 1's, so no carries possible
  - |C| = k
  - at least one $x_i$ must contribute to sum for $e_j$

| | | $e_1$ | $e_2$ | $e_3$ | $e_4$ | $e_5$ | $e_6$ | decimal |
|---|---|---|---|---|---|---|---|---|
| $x_1$ | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 5,184 |
| $x_2$ | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 4,356 |
| $x_3$ | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 4,116 |
| $x_4$ | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 4,161 |
| $x_5$ | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 5,393 |
| $y_1$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1,024 |
| $y_2$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 256 |
| $y_3$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 64 |
| $y_4$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 16 |
| $y_5$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 4 |
| $y_6$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| t | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 15,018 |
| | k | | | | | | | |




68

## Partition

SUBSET-SUM: Given a set X of integers and a target integer t, is there a subset S ⊆ X whose elements sum to exactly t.

PARTITION: Given a set X of integers, is there a subset S ⊆ X such that $\sum_{a \in S} a = \sum_{a \in X \setminus S} a$.

Claim. SUBSET-SUM $\le_P$ PARTITION.
Proof. Let (X, t) be an instance of SUBSET-SUM.
- Define W to be sum of integers in X: $W = \sum_{a \in X} a$.
- Create instance of PARTITION: X' = X ∪ {2W - t} ∪ {W + t}.
- SUBSET-SUM instance is yes if and only if PARTITION instance is.
  - in any partition of X'
    - Each half of partition sums to 2W.
    - Two new elements can't be in same partition.
    - Discard new elements ⇒ subset of X that sums to t.

69

**Theorem 2** *Greedy outputs an independent set $S$ such that $|S| \geq n/(\Delta + 1)$ where $\Delta$ is the maximum degree of any node in the graph.*

GREEDY($G$):
$S \leftarrow \emptyset$
While $G$ is not empty do
  Let $v$ be a node of minimum degree in $G$
  $S \leftarrow S \cup \{v\}$
  Remove $v$ and its neighbors from $G$
end while
Output $S$

**Proof:** We upper bound the number of nodes in $V \setminus S$ as follows. A node $u$ is in $V \setminus S$ because it is removed as a neighbor of some node $v \in S$ when Greedy added $v$ to $S$. Charge $u$ to $v$. A node $v \in S$ can be charged at most $\Delta$ times since it has at most $\Delta$ neighbors. Hence we have that $|V \setminus S| \leq \Delta|S|$. Since every node is either in $S$ or $V \setminus S$ we have $|S| + |V \setminus S| = n$ and therefore $(\Delta + 1)|S| \geq n$ which implies that $|S| \geq n/(\Delta + 1)$. □

Since the maximum independent set size in a graph is $n$ we obtain the following.

**Corollary 3** *Greedy gives a $\frac{1}{\Delta+1}$-approximation for (unweighted)* MIS *in graphs of degree at most $\Delta$.*

**LP Relaxation:** One can formulate a simple linear-programming relaxation for the (weighted) MIS problem where we have a variable $x(v)$ for each node $v \in V$ indicating whether $v$ is chosen in the independent set or not. We have constraints which state that for each edge $(u, v)$ only one of $u$ or $v$ can be chosen.

$$
\begin{aligned}
\text{maximize } & \sum_{v \in V} w(v)x(v) \\
\text{subject to } & x(u) + x(v) \leq 1 \qquad (u, v) \in E \\
& x(v) \in [0, 1] \qquad v \in V
\end{aligned}
$$

Although the above is a valid integer programming relaxation of MIS when the variabels are constrained to be in $\{0, 1\}$, it is not a particularly useful formulation for the following simple reason.

**Claim 4** *For any graph the optimum value of the above LP relaxation is at least $w(V)/2$. In particular, for the unweighted case it is at least $n/2$.*

Linear Programming:
Min/Max a linear objective subject to linear constraints:
Min: $E\ c_i x_i$ Max: $Ec_i x_i$
s.t $Ax >= b$ $Ax <= b$
In most cases, $x >=0$ because they're nonnegative.
Any maximization/minimization problem can be written as LP.
Integer LP programming is NP hard, but it generalizes every problem.
Duality: change a max to min and vice versa

| Primal: | Dual: |
|---|---|
| Min cx | max by |
| s.t. $Ax >= b$ | s.t $A^T y <= c$ (transpose of A) |

NP
NP means a problem that we do not know of a polynomial time algorithm, and we cannot prove one exists.
How to prove a problem is NP hard:

1. If we know a solution to the problem, then we can verify the solution is right/wrong in polynomial time. Thus, we need to prove that if a solution is provided, we have an algorithm that validates the solution and prove its polynomial time.
2. Identify a known NP hard problem x, and do a reduction to the current problem y, and we need to prove that x is polynomial time reducible to y. $x <=_p y$. We should also prove that $y <=_p x$.
3. Proving $x <=_p y$. first, do a reduction from x to y to prove the reduction is valid. Next, prove that the reduction is polynomial time.

Independent Set: Given a graph G= (V,E) we say that a set of nodes $S \subseteq V$ is independent if no two nodes in S are joined by an edge, i.e., nodes in S are not adjacent. IS at least k (maximization)
Vertex Cover: Given a graph G=(V,E), we say that a set of nodes $S \subseteq V$ is a Vertex Cover if every edge $e \in E$ has at least one end in S, i.e., S covers all edges. VC is at most k (minimization problem).
Proof IS $<=_p$ VC
- First, suppose that S is an independent set.
- Let e = (u, v) be an arbitrary edge in G.
- Since S is independent, it cannot be the case that both u and v are in S as that would contradict the claim that S is an independent set.
- Therefore, one of the endpoints of L must lie in the set V-S.
- Therefore, since S is an independent set, it follows that this must be true $\forall e \in G$. i.e., every edge has at least one end in V-S . By definition, V-S is a vertex cover.
- Suppose V-S is a vertex cover.
- Consider any two nodes u and $v \in S$.
- If u and v were joined by an edge, then not both ends of the edge would lie in $V - S$, contradicts our assumption that $V - S$ is a Vertex Cover.
- No two nodes in S are joined by an edge.
- So, S is an independent set. We can conclude that IS and VC are closely related to each other.

Input: given a set X of n Boolean variables x1, x2, .., xn each can take the value 0 or 1 (equivalently to false or true).
A clause is a disjunction of distinct terms where every term contains the variable xi or xi'
F is a formula consists of conjunction of clauses.
E.g. $F = (x1 \vee x2 \vee x3') \wedge (x4 \vee x5' \vee x1) \wedge (x3' \vee x4' \vee x6)$
F is satisfiable if we can assign truth values to variables (not literals -- a var or it's negation) to make the entire formula true. In this example (x1 = 1 and x2 = x3 = x4 = x5 = x6 = 0) • Satisfiability problem: given a set of clauses C, C1,…, Ck , over a set of X = {x1, x2,…,xn}, is there a satisfying truth assignment?
3SAT is each clause is restricted to EXACTLY 3 literals but we can have any number of clauses
Sat to 3SAT conversion:
One var, 2 unknown: $\{x \vee z1 \vee z2\}$, $\{x \vee z1 \vee z2'\}$, $\{x \vee z1' \vee z2\}$, and $\{x \vee z1' \vee z2'\}$
Two var, 1 unknown: $\{x1 \vee x2 \vee z\}$ and $\{x1 \vee x2 \vee z'\}$
3 var, 0 unknown (3-Sat): $\{x1 \vee x2 \vee x3\}$.
4+ var:
Let Ci be equal to $\{x1 \vee x2 \vee . . . \vee xk\}$. We create $k - 3$ new variables and $k - 2$ new clauses in a chain where for $2 \le j \le j - 3$, $C_{i,j} = \{z_{i,j-1} \vee x_{j+1} \vee z_{i,j}'\}$, $C_{i,1} = \{x1 \vee x2 \vee z_{i,1};\}$, and $C_{i,k-2} = \{z_{i,k-3} \vee x_{k-1} \vee x_k\}$ If none of the original literals in Ci is true, then there are not enough free variables to be able to satisfy all of the new subclasses. If you satisfy $C_{i,1}$ by setting $z_{i,1}$ to false, it would require $z_{1,2}$ = false and so on until $C_{i,k-2}$ cannot be satisfied and thus the clause cannot be satisfied. However, if any of the single literal xi is

L = a+b+
E = {a, b, _} #alphabets
Q = {s0, ACCEPT, REJECT, s1, s2} #stats
(a, s0) -> (a, s1, 'move right')
(b, s0) -> (b, REJECT, stay);
(_, s0) -> (_, REJECT, stay);
(a, s1) -> (a, s1, move right);
(b, s1) -> (b, s2, move right);
(_, s1) -> (_, REJECT, stay);
(a, s2) -> (a, REJECT, stay);
(b, s2) -> (b, s2, move right);
(_, s2) -> (_, accept, stay);

L = a*b*
E = {a, b, _} #alphabets
Q = {s0, ACCEPT, REJECT, s1, s2} #stats
(a, s0) -> (a, s1, move right)
(b, s0) -> (b, s2, move right)
(_, s0) -> (_, accept, stay)
(a, s1) -> (a, s1, move right)
(b, s1) -> (b, s2, move right)
(_, s1) -> (_, accept, stay)
(a, s2) -> (a, reject, stay)
(b, s2) -> (b, s2, move right)
(_, s2) -> (_, accept, stay)
The infinite loop in a TM example:
{a, si} -> (a, sj, 'move right')
{b, sj} -> (b, si, "move left");
The number of transition entries for a TM equals to |E| x |Q - {accept, reject}|

Weighted Set Cover (NP Hard):
U -> universe of size m
$S_1$,…,$S_k$ are subsets of U, such that $U\ S_i = U$.
Each $s_i$ has a weight $w_i > 0$.
Subset of {$s_1$, .., $s_k$} is a "set cover" if the union of the elements of the subset = U.
Goal: Find a set cover with the minimum weight.
E $W_{alpha_i}$ is minimized.
Greedy rule: Chose the next $s_i$ that minimizes $w_i/|s_i|$ where $|s_i|$ is num of elems in $S_i$
Algo:
T is the set of uncovered elements. Pick an $s_i$ that minimized ($w_i/|s_i$ intersect T|). We get a O(log(n)) approximation ratio.
Charging Scheme: How much do you charge for elements covered?
Claim: For any set $s_i$ total of charges assigned to elements of $s_i <= w_i(1+ .5 + … + 1/|s_i|)$.
Approximation Proof:
Let $S_1$, $S_2$, $S_3$) be the optimal set cover.
Therefore, the cost of adding $S_1 <= w_1*H_n$, $S_3 <= w_3*H_n$, $S_3 <= w_3*H_n$. Therefore, the total cost $<= (w_1 + w_2 + w_3) * H_n$. Therefore, the total cost = $w*H_n$
Converting from Vertex Cover to Set Cover. U is the list of all edges, $S_i$ is the list of edges incident on vertex i.
We can get an approximation of $H_d$ where d is the max degree of the graph.
2-approx for weighted vertex cover.
Maintain a non negative charge $C_e$ for each edge e. Initialize $C_e = 0$ for all edges. Maintain invariant that $E\ C_{(u,v)} <= w_u$ for every u in V.
The charges induce a coloring of the vertices if inequality for a vertex is an equality. If it is a tight inequality, it is red, not tight, it is blue.
Which there is an edge that is blue, increase C(u,v) until at least one of (u,v) becomes red. Return the set of red vertices. Let u* be an optimal vertex cover with w(u*) = w*. Let U be the output of our algorithm, w(U) <= 2w*
Claim: w(u*) >= E $c_e$ = sum of all edge charges.
$w_i$>= total remaining charge on edges adjacent to :

Clique: IS $<=_p$ clique. A clique is a subset S of V such that for every two nodes u, v in S, there exists an edge from u to v. The reduction is by creating a complement graph G', which has same vertices but opposite edges. E' = V X V – E.

Approximation Algorithm:
[our answer] <= alpha x [correct answer] where alpha >= 1;
A Load Balancing Problem (NP – HARD)
M machines, N Jobs. Each job has a non negative weight $w_i$.
Goal: Minimize the max load of any one machine.
First version of input is non sorted.
L* is the optimal maximum load.
Claims: w<= L* and L <= L*
Total load = M*L. Therefore average load is M*L/M <= L*
Claims: 2L* >= w + L by adding the above two claims
This holds at every step, which means our answer <= 2L*
Second version of input is sorted in descending order of weights
Add a weight w to a machine of load L
Case1:: L = 0 -> L+w = w <= L*
Case2: L>0, at least m+1 jobs have the weight >= w.
L* >= 2w, w <= L*/2
L + w <= L* + L*/2 <= 1.5 L*

Therefore, E $w_i$ >= E total.
w(u*) >= E $c_e$ where e belongs to E.
Claim: w(u) <= 2 * E $c_e$.
= 2* Total of all charges
Consider an edge (u,v), fill in the buckets with c(u,v)$ in each bucket. Observe that the total $ in bag at u = $w_u$ if u E U. Thus, the total $ distributed = E $w_u$ = w(u) to vertices in U.
2*Total charges = Total distributed. Thus, 2*E$c_e$ >= total $ dist = E $w_u$ = W(u).

$A_{tm}$ = {(M,w), M is a TM and M accepts w}
Assum $A_{tm}$ is decidable, and thus we construct H a decider for $A_{tm}$ such that
H (M,w) = {accept if M accepts w, and reject if M does not accept w}
Construct a new TM D with H as a subroutine. It feeds a TM as the input.
D = "On input <M>, where M is a TM:

1. Run H on input <M, <M>>
2. Output the opposite of what H outputs, that is if H accepts, reject, and if H rejects, accept.

Call D with itself as an input. D<D> = {accept if D does not accept D, reject if D accepts. This is a contradiction..
If the language L is decidable, then its complement L bar is also decidable. If L is decidable, there exists a TM that can detect L and another TM that can detect L bar.
Atm = <M,w> ; M is a turing maching, w is a string
M -> accepts if w belongs to L
-> reject if w does not belong to L
If L is deciable then L' is also decidable. i.e L' has a turing m/c M' such that M' accept w if w belong L' and rejects if w does not belong to L'.