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\*Ec\_e >= total $ dist = E w\_u = W(u).

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\_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 i.

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\*

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'.

You can construct M' using M as a sub-routine.

M' => accepts  if M reject w

   => rejects  if M accepts w

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}|

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.

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 ∨ x2 ∨ x3’ ) ∧ (x4∨ x5’ ∨ x1 ) ∧ (x3’ ∨ x4’ ∨ 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 ∨ z1 ∨ z2}, {x ∨ z1 ∨ z2’}, {x ∨ z1’ ∨ z2}, and {x ∨ z 1’ ∨ z2’}

Two var, 1 unknown: {x1 ∨ x2 ∨ z} and {x1 ∨ x2 ∨ z’}

3 var, 0 unknown (3-Sat): {x1 ∨x2 ∨x3}.

4+ var:

Let Ci be equal to {x1 ∨x2 ∨. . .∨xk}. We create k − 3 new variables and k − 2 new clauses in a chain where for 2 ≤ j ≤ j − 3, Ci,j = {zi,j−1 ∨ xj+1 ∨ z i,j’}, Ci,1 = {x1 ∨ x2 ∨ z i,1; }, and Ci,k−2 = {zi,k−3 ∨ xk−1 ∨ xk} 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 Ci,1 by setting zi,1 to false, it would require z1,2 = false and so on until Ci,k−2 cannot be satisfied and thus the clause cannot be satisfied. However, if any of the single literal xi is equal to true, then we have k − 3 free variables and k − 3 remaining clauses, and thus each of the clauses can be satisfied. Thus, Ci can be satisfied.

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 ∀e∈ 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 ∈ 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.

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 ⊆ 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 ⊆ V is a Vertex Cover if every edge e ∈ E has at least one end in S, i.e., S covers all edges. VC is at most k (minimization problem).

Linear Programming:

Min/Max a linear objective subject to linear constraints:

Min: E c\_i x\_i Max: Ec\_ix\_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)