Load Balancing Problem

(Approximation Example)

Input:

m resources with equal processing power

n jobs, where job j takes t_j to process

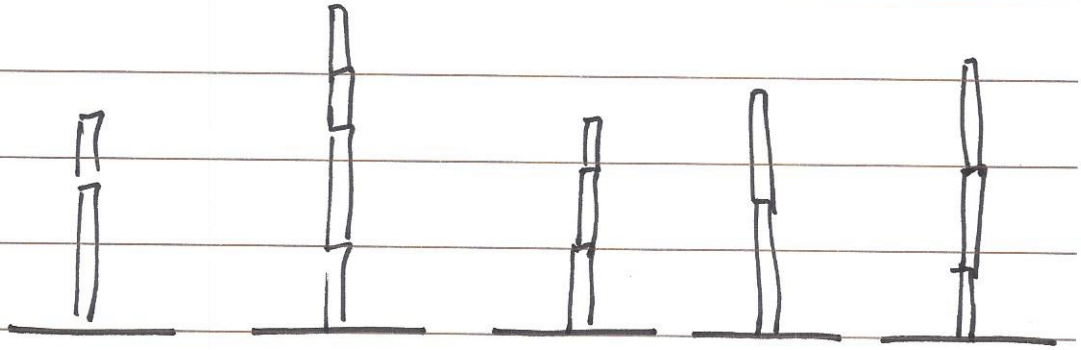
objective:

Assignment of jobs to resources such that the maximum load on any machine is minimized.

Notation:

T_i : load on machine/resource i

T^* : value of the optimal solution



Greedy Balancing

Improved approximation to greedy balancing

Initially sort jobs in decreasing order of length then use same greedy balancing

Vertex Cover Problem
(Approximation Example)

Problem Statement: Find the smallest vertex cover in graph G.

A 2-approximation algorithm to the vertex Cover problem:

```
Start with S = Null
While S is not a vertex cover
    Select an edge e not covered by S
    Add both ends of e to S
EndWhile
```

Why is this a 2-approximation?

Question: Since Independent Set $\leq_p$ Vertex Cover,

Can I use our 2-approximation algorithm for vertex cover to find a .5-approximation to independent set?

Theorem: Unless P=NP, there is no $\frac{1}{n^{1-\epsilon}}$ approximation for the Maximum Independent Set problem for any $\epsilon > 0$ where n is the no. of nodes in the graph.

Question: Since Vertex Cover $\leq_p$ Set Cover

Can I use a 2-approximation algorithm for Set Cover to find a 2-approximation to Vertex Cover?

MAX-3SAT

Given a set of clauses of length 3, find a truth assignment that satisfies the largest number of clauses.

A .5-approximation to the MAX-3SAT problem:

- Set everything to true
  If less than 50% of clauses evaluate to true, then
- Set everything to False

More sophisticated approximation methods can get to within a factor of 8/9 of the optimal sol.

System of Linear Equations

$$[A][X] = [B]$$

Linear Programming

$$[A][X] \leq [B]$$

Objective function: $[C^T][X]$

Goal: Maximize the objective function subject to the above constraints

Linear Programming Standard Form

- All constraints are of the form $\leq$
- All variables are non-negative
- Objective function is maximized.

Ex.: $x_1 - x_2 \geq 0$

$x_1 \geq 0$

$x_2 \geq 0$

$x_1 + x_2 \leq 4$

Weighted Vertex Cover Problem

For $G=(V,E)$, $S \subseteq V$ is a vertex cover set such that each edge has at least one end in S

Also, $w_i \geq 0$ for each $i \in V$

So, the total weight of the set $= W(S) = \sum_{i \in S} w_i$

Objective: Minimize $W(S)$

- Linear Programming
  Continuous Variables
- Integer Programming
  Discrete variables
- Mixed Integer Programming
  Both Cont. & Discrete
- Non-Linear Programming
  Non Linear Constraints or Objective Functions

To find an approximate solution using LP, drop the requirement that $x_i \in \{0,1\}$ and solve the LP in polynomial time to find $\{x_i^*\}$ between 0 & 1.

$$W_{LP} = \sum_i w_i x_i^*$$

Assume S' is the optimal vertex cover set and $W(S')$ is the weight of the opt. solution

Maxflow Problem

Variables are flows over edges

objective function: Maximize $\sum_{e \text{ out of } S} f(e)$

Subject to

$0 \leq f(e) \leq c_e$ for each edge $e \in E$

$\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = 0$ for $v \in V$ except for s & t

Shortest Path using LP

Problem Statement:

Find shortest path from s to t.

Discussion 12

1. The *bin packing* problem is as follows. You have an infinite supply of bins, each of which can hold M maximum weight. You also have n objects, each of which has a (possibly distinct) weight w_i (any given w_i is at most M). Our goal is to partition the objects into bins, such that no bin holds more than M total weight, and that we use as few bins as possible. This problem in general is **NP**-hard.

Give a 2-approximation to the *bin packing* problem. That is, give an algorithm that will compute a valid partitioning of objects into bins, such that no bin holds more than M weight, and our algorithm uses at most twice as many bins as the optimal solution. Prove that the approximation ratio of your algorithm is two.

2. Suppose you are given a set of positive integers A: $a_1\ a_2\ \ldots\ a_n$ and a positive integer B. A subset $S \subseteq A$ is called *feasible* if the sum of the numbers in S does not exceed B.

The sum of the numbers in S will be called the *total sum* of S. You would like to select a feasible subset S of A whose total sum is as large as possible.

Example: If A = {8, 2, 4} and B = 11 then the optimal solution is the subset S = {8, 2}.

Give a linear-time algorithm for this problem that finds a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S' \subseteq A$. Prove that your algorithm achieves this guarantee.
You may assume that each $a_i \leq B$.

3. A cargo plane can carry a maximum weight of 100 tons and a maximum volume of 60 cubic meters. There are three materials to be transported, and the cargo company may choose to carry any amount of each, up to the maximum available limits given below.

- Material 1 has density 2 tons/cubic meter, maximum available amount 40 cubic meters, and revenue $1,000 per cubic meter.
- Material 2 has density 1 ton/cubic meter, maximum available amount 30 cubic meters, and revenue $1,200 per cubic meter.
- Material 3 has density 3 tons/cubic meter, maximum available amount 20 cubic meters, and revenue $12,000 per cubic meter.

Write a linear program that optimizes revenue within the constraints. You do not need to solve the linear program.

4. Recall the 0/1 knapsack problem:

Input: n items, where item i has profit p_i and weight w_i, and knapsack size is W.

Output: A subset of the items that maximizes profit such that the total weight of the set ≤ W.

You may also assume that all $p_i \geq 0$, and $0 < w_i \leq W$.

We have created the following greedy approximation algorithm for 0/1 knapsack:

Sort items according to decreasing p_i/w_i (breaking ties arbitrarily).

While we can add more items to the knapsack, pick items in this order.

Show that this approximation algorithm has no nonzero constant approximation ratio.

In other words, if the value of the optimal solution is P*, prove that there is no constant ρ ($1 > \rho > 0$), where we can guarantee our greedy algorithm to achieve an approximate solution with total profit ρ P*.

A set of n space stations need your help in building a radar system to track spaceships traveling between them. The i^{th} space station is located in 3D space at coordinates (x_i, y_i, z_i). The space stations never move. Each space station i will have a radar with power r_i, where r_i is to be determined. You want to figure how powerful to make each space station's radar transmitter, so that whenever any spaceship travels in a straight line from one space station to another, it will always be in radar range of either the first space station (its origin) or the second space station (its destination). A radar with power r is capable of tracking spaceships anywhere in the sphere with radius r centered at itself. Thus, a spaceship is within radar range through its trip from space station i to space station j if every point along the line from (x_i, y_i, z_i) to (x_j, y_j, z_j) falls within either the sphere of radius r_i centered at (x_i, y_i, z_i) or the sphere of radius r_j centered at (x_j, y_j, z_j). The cost of each radar transmitter is proportional to its power, and you want to minimize the total cost of all of the radar transmitters. You are given all of the (x_1, y_1, z_1), ... ,(x_n, y_n, z_n) values, and your job is to choose values for $r_1,...,r_n$. Express this problem as a linear program.

Randomized Algorithms

Two general types of randomizations

1- Algorithm relies on random nature of input

2- Algorithm behaves randomly

Two types of Results

1- Some randomized algorithms always produce correct results. Randomization only affects the run time.

2- Some randomized algorithms may produce incorrect solutions, but we try to bound the probability of such an incorrect solutions.

Ex 1: Problem Statement:

Find the median in an unsorted array.

Another Approach

If looking for the median, we then only need to look into either the right half or the left half. (Not necessarily equal halves)

Solution:

Use the random nature of data/input to find the median:

Divide: Pick a pivot value (last term in the array) and split the array in two.
Terms in left half are all less than the pivot value
Terms in right half are all greater than the pivot value.

Conquer:

if x is the median term we are looking for return it
else if the median is in the left half then explore the left half recursively
else explore the right half recursively
end if

A Simplistic Analysis of Runtime

Assume array size is only reduced by 10% each time.

$$T(n) = cn + (9/10)cn + (9/10)^2 cn + \ldots$$

Ex. 2: Global Min-Cut Problem

Given a connected unweighted multigraph, partition vertices into two disjoint sets with the minimum no. of edges connecting the two set.

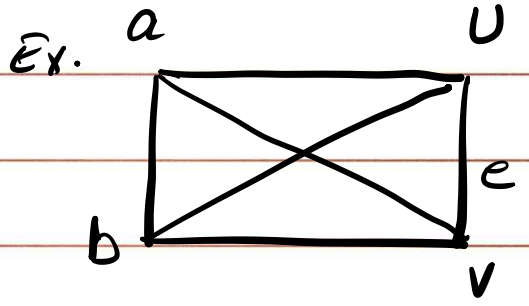
Deterministic solution

- Pick any vertex as a source
- For each other vertex in V
  Run Max Flow and find min cut.
- Return smallest Min Cut.

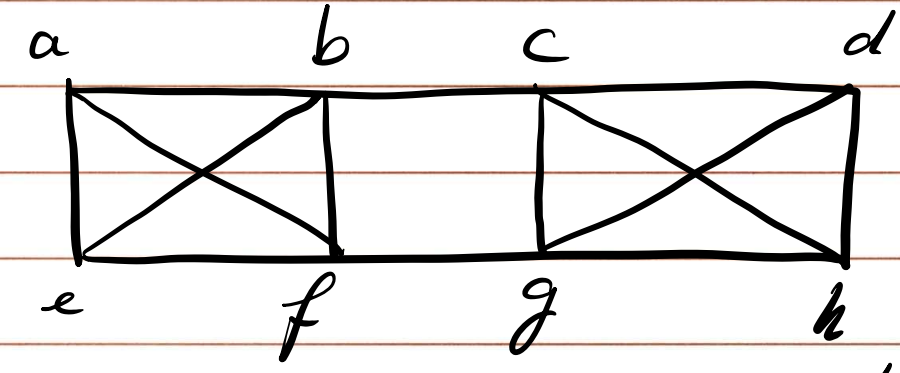
Randomize Solutions Using Edge Contractions

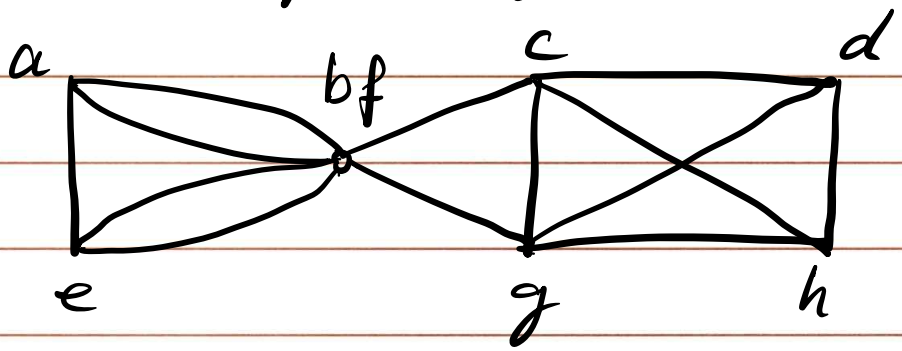
Edge Contraction: Contraction of an edge e with endpoints U & V is the replacement of U and V with a single vertex such that edges incident to the new vertex are the edges (other than e) that were incident on U or V.

Ex.

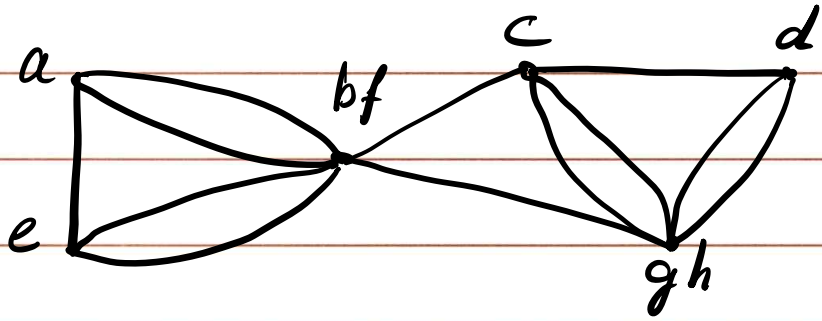


Find Global Min-Cut in the graph below:

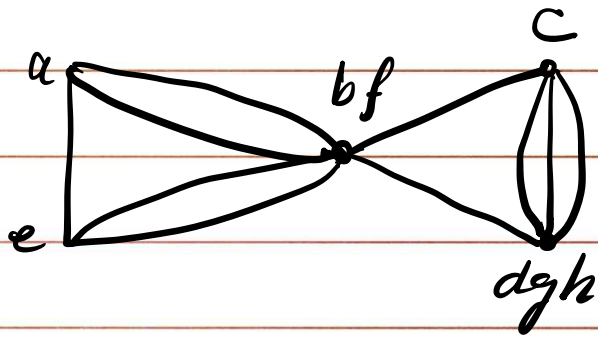




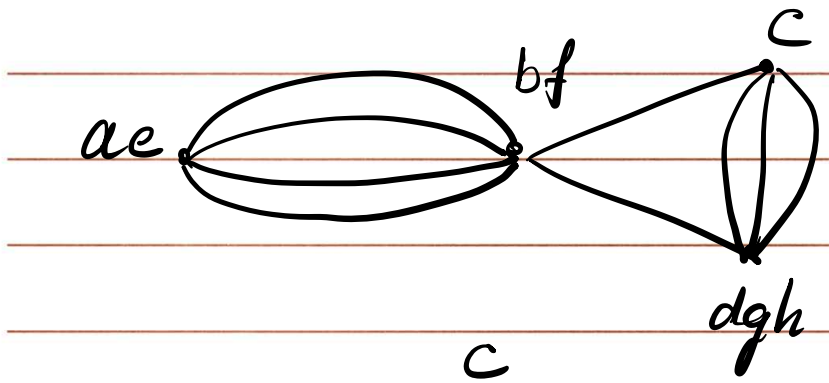
Contract edge bf



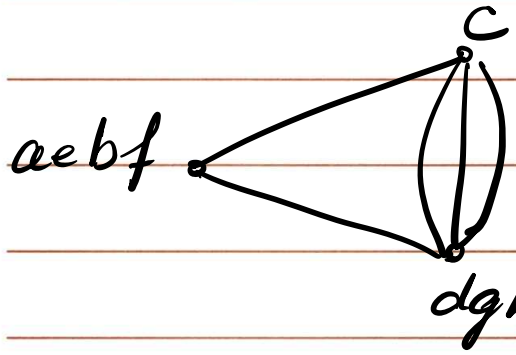
Contract gh



Contract d(gh)



Contract ae



Contract (ae)(bf)

Contract c(dgh)

Randomized Algorithm

While there are more than 2 nodes left
pick an edge at random
and contract the edge
(Remove any self loops)
end while

For the two remaining nodes U_1 & U_2
Set V_1 = {nodes that went into U_1}
V_2 = { " " " " U_2}

Analysis

Assume that a graph has a single min cut C. If we never contract edges from that min cut, then the algorithm will end up with the correct Min Cut.

How likely is this?

Say size of min cut is k.
So each edge must have at least degree k.

$\Rightarrow$ Total no. of edges in $G \geq kn/2$

So, the probability that the 1st contracted edge is Not on our min cut is at least $1 - \frac{k}{kn/2}$

The probability that all contracted edges are NOT on our min cut is at least

$$\left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdot\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{3}\right)$$

$$= \Omega\left(1/n^2\right)$$

The probability of the algorithm failing N times is

$$\left(1-1/n^2\right)^N$$

If we choose $N = 100n^2$ we can make it close to zero! Since $\left(1-1/n^2\right)^{100n^2} \leq e^{-100}$

$$\left(\lim_{x\to\infty}\left(1-1/x\right)^{ax} = e^{-ax}\right)$$