Approximation Algorithms

What do you do when a problem is NP-complete?

- or, when the "polynomial time solution" is impractically slow?
- assume input is random, do "expected performance." Eg, Hamiltonian path in all graphs. Problem: agreeing on good distribution.
- run a nonpolynomial (hopefully only slightly) algorithms such as branch and bound. Usually no proven bound on runtime, but sometime can.
- settle for a *heuristic*, but prove it does *well enough* (our focus)

Definitions:

- optimization problem, instances I, solutions $S(I)$ with values $f : S(I) \rightarrow R$
- maximization/minimization: find solution in $S(I)$ maximizing/minimizing f
- called OPT(I)
- eg bin-packing. instance is set of $s_i \in 0, 1$, partition so no subset exceeds 1

Techincal assumptions we'll often make:

- assumption: all inputs and range of f are integers/rationals (can't represent reals, and allows, eg, LP, binary search).
- assumption $f(\sigma)$ is a polynomial size (num bits) number (else output takes too long)
- look for polytime in bit complexity

NP-hardness

- optimization NP-hard if can reduce an NP-hard decision problem to it
- (eg, problem of "is opt solution to this instance $\leq k$?")
- but use more general notion of turing-reducibility (GJ).

Approximation algorithm:

- any algorithm that gives a feasible answer
- eg, each item in own bin.
- of course, want *good* algorithm. How measure?

Absolute Approximations

Definition: k-abs approx if on any I, have $|A(I) - OPT(I)| \leq k$
Example: planar graph coloring.

- NP-complete to decide if 3 colorable
- know 4-colorable
- easy to 5-color
- Ditto for edge coloring: Vizing's theorem says opt is Δ or (constructive) $\Delta + 1$

Known only for trivial cases, where opt is bounded by a constant.
Often, can show impossible by "scaling" the problem.

- EG knapsack.
  - define profits p_i, sizes s_i, sack B
  - wlog, integers.
  - suppose have k-absolute
  - multiply all p_i by $k + 1$, solve, scale down.
- EG independent set (clique)
  - $k + 1$ copies of G

Relative Approximation

Definitions:

- An *α-optimum* solution has value at most α times optimum for minimization, at least $1/\alpha$ times optimum for minimization.
- an algorithm has *approximation ratio* α if on any input, it outputs an α-approximate feasilbe solution.
- called an *α-approximation algorithm*

How do we prove algorithms have relative approximations?

- Can't describe opt, so can't compare to it
- instead, comparison to computable lower bounds.

Greedy Algorithms

Do obvious thing at each step.

- Hard part is proving it works.
- Usually, by attention to right upper/lower bound.

Max cut

- Upper bound trivial
- Max-cut greedy.

Min-diameter clustering?

- Gonzales' algorithm.
- Distances to existing centers keep dropping
- Suppose after k chosen, farthest remaining is distance d
- Then OPT $\geq d$
  - $k+1$ mutually-distance-d points
  - some must share a cluster
- Now assign each point to closest center
- Max distance from center (radius) is d
- So max diameter is $2d$
- 2-approx.

Set cover

- n items
- OPT $= k$
- At each step, can still cover remainder with k sets
- So can cover $1/k$ of remainder

Vertex cover:

- define problem
- suppose repeatedly pick any uncovered edge and cover: no approx ratio
- suppose pick uncovered edge and cover both sides: 2-approx
- sometime, need to be extra greedy

- Explicit attention to where lower bound is coming from—lower bound informs algorithm.

Graham's rule for $P||C_{\max}$ is a $2 - \frac{1}{m}$ approximation algorithm

- explain problem: m machines, n jobs with proc times p_j, min proc time.
- can also think of minimizing max load of continuously running jobs
- use a *greedy algorithm* to solve
- proof by comparison to lower bounds
- first lower bound: average load: $\mathrm{OPT} \geq \frac{1}{m} \sum p_j$
- second lower bound: $\mathrm{OPT} \geq \max p_j$
- suppose M_1 has max runtime L at end
- suppose j was last job added to M_1
- then before, M_1 had load $L - p_j$ which was minimum
- so $\sum p_i \geq m(L - p_j) + p_j$
- so $\mathrm{OPT} \geq L + (1 - \frac{1}{m})p_j$
- so $L \leq \mathrm{OPT} + (1 - \frac{1}{m})p_j \leq (2 - \frac{1}{m})\mathrm{OPT}$

Notice:

- this algorithm is *online*, competitive ratio $2 - \frac{1}{m}$
- we have no idea of optimum schedule; just used lower bounds.
- we used a greedy strategy
- tight bound: consider $m(m-1)$ size-1 jobs, one size-m job
- where was problem? Last job might be big
- LPT achieves 4/3, but not online
- newer online algs achieve 1.8 or so.

Approximation Schemes

So far, we've seen various constant-factor approximations.

- WHat is *best* constant achievable?
- Lower bounds: APX-hardness/Max-SNP

An *approximation scheme* is a family of algorithms A_ϵ such that

- each algorithm polytime
- A_ϵ achieve $1 + \epsilon$ approx

But note: runtime might be awful as function of ϵ

FPAS, Pseudopolynomial algorithms

Knapsack

- Dynamic program for bounded profits
- $B(j, s)$ = best subset of jobs $1, \ldots, j$ of total size $\leq s$.
- rounding
  - Let opt be P.
  - Scale prices to $\lfloor (n/\epsilon P)p_i \rfloor$
  - New opt is it least $n/\epsilon - n = (1-\epsilon)n/\epsilon$
  - So find solution within $1-\epsilon$ of original opt
  - But table size polynomial
- did this prove $P = NP$? No
- recall pseudopoly algorithms

pseudopoly gives FPAS; converse almost true

- Knapsack is only *weakly* NP-hard
- strong NP-hardness (define) means no pseudo-poly

Enumeration

More powerful idea: k-enumeration

- Return to $P||C_{\max}$
- Schedule k largest jobs optimally
- scheduling remainder greedily
- analysis: note $A(I) \leq OPT(I) + p_{k+1}$
  - Consider job with max c_j
  - If one of k largest, done and at opt
  - Else, was assigned to min load machine, so $c_j \leq OPT + p_j \leq OPT + p_{k+1}$
  - so done if p_{k+1} small
  - but note $OPT(I) \geq (k/m)p_{k+1}$
  - deduce $A(I) \leq (1 + m/k)OPT(I)$.
  - So, for fixed m, can get any desired approximation ratio

Scheduling any number of machines

- Combine enumeration and rounding
- Suppose only k job sizes
  - Vector of "number of each type" on a given machine—gives "machine type"
  - Only n^k distinct vectors/machine types
  - So need to find how many of each machine type.
  - Use dynamic program:
    * enumerate all job profiles that can be completed by j machines in time T
    * In set if profile is sum of $j-1$ machine profile and 1-machine profile
  - Works because only poly many job profiles.
- Use *rounding* to make few important job types
  - Guess OPT T to with ϵ (by binary search)
  - All jobs of size exceeding ϵT are "large"
  - Round each up to next power of $(1+\epsilon)$
  - Only $O(1/\epsilon \ln 1/\epsilon)$ large types
  - Solve optimally
  - Greedy schedule remainder
    * If last job is large, are optimal for rounded problem so with ϵ of opt
    * If last job small, greedy analysis showes we are within ϵ of opt.

Relaxations

TSP

- Requiring tour: no approximation (finding hamiltonian path NP-hard)
- Undirected Metric: MST relaxation 2-approx, christofides
- Directed: Cycle cover relaxation

LP relaxations

Three steps

- write integer linear program
- relax
- round

Vertex cover.

MAX SAT

Define.

- literals
- clauses
- NP-complete

random setting

- achieve $1 - 2^{-k}$
- very nice for large k, but only 1/2 for $k = 1$

LP

$$\begin{aligned} \max \quad & \sum z_j \\ \sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_1) \quad \geq \quad & z_j \end{aligned}$$

Analysis

- $\beta_k = 1 - (1 - 1/k)^k$. values $1, 3/4, .704, \ldots$
- Random round y_i
- Lemma: k-literal clause sat w/pr at least $\beta_k \hat{z}_j$.
- proof:
  - assume all positive literals.
  - prob $1 - \prod(1 - y_i)$
  - maximize when all $y_i = \hat{z}_j/k$.
  - Show $1 - (1 - \hat{z}/k)^k \geq \beta_k \hat{z}_k$.
  - check at $z = 0, 1$
- Result: $(1 - 1/e)$ approximation (convergence of $(1 - 1/k)^k$)
- much better for small k: i.e. 1-approx for $k = 1$

LP good for small clauses, random for large.

- Better: try both methods.
- n_1, n_2 number in both methods
- Show $(n_1 + n_2)/2 \geq (3/4) \sum \hat{z}_j$
- $n_1 \geq \sum_{C_j \in S^k} (1 - 2^{-k}) \hat{z}_j$
- $n_2 \geq \sum \beta_k \hat{z}_j$
- $n_1 + n_2 \geq \sum (1 - 2^{-k} + \beta_k) \hat{z}_j \geq \sum \frac{3}{2} \hat{z}_j$

0.1 Chernoff-bound rounding

Set cover.
Theorem:

- Let X_i poisson (ie independent 0/1) trials, $E[\sum X_i] = \mu$

$$\Pr[X > (1+\epsilon)\mu] < \left[\frac{e^\epsilon}{(1+\epsilon)^{(1+\epsilon)}}\right]^\mu.$$

- note independent of n, exponential in μ.

Proof.

- For any $t > 0$,

$$\begin{aligned}\Pr[X > (1+\epsilon)\mu] &= \Pr[\exp(tX) > \exp(t(1+\epsilon)\mu)] \\ &< \frac{E[\exp(tX)]}{\exp(t(1+\epsilon)\mu)}\end{aligned}$$

- Use independence.

$$\begin{aligned}E[\exp(tX)] &= \prod E[\exp(tX_i)] \\ E[\exp(tX_i)] &= p_i e^t + (1-p_i) \\ &= 1 + p_i(e^t - 1) \\ &\leq \exp(p_i(e^t-1))\end{aligned}$$

  $\prod \exp(p_i(e^t-1)) = \exp(\mu(e^t-1))$

- So overall bound is

$$\frac{\exp((e^t-1)\mu)}{\exp(t(1+\epsilon)\mu)}$$

  True for any t. To minimize, plug in $t = \ln(1+\epsilon)$.

- Simpler bounds:
  - less than $e^{-\mu\epsilon^2/3}$ for $\epsilon < 1$
  - less than $e^{-\mu\epsilon^2/4}$ for $\epsilon < 2e-1$.
  - Less than $2^{-(1+\epsilon)\mu}$ for larger ϵ.

- By same argument on $\exp(-tX)$,

$$\Pr[X < (1-\epsilon)\mu] < \left[\frac{e^{-\epsilon}}{(1-\epsilon)^{(1-\epsilon)}}\right]^\mu$$

  bound by $e^{-\epsilon^2/2}$.

Basic application:

- $cn \log n$ balls in c bins.
- max matches average
- a fortiori for n balss in n bins

General observations:

- Bound trails off when $\epsilon \approx 1/\sqrt{\mu}$, ie absolute error $\sqrt{\mu}$
- no surprise, since standard deviation is around μ (recall chebyshev)
- If $\mu = \Omega(\log n)$, probability of constant ϵ deviation is $O(1/n)$, Useful if polynomial number of events.
- Note similarito to Gaussian distribution.
- **Generalizes:** bound applies to any vars distributed in range $[0, 1]$.

Zillions of Chernoff applications.
Wiring.

- multicommodity flow relaxation
- chernoff bound
- union bound