L03 Greedy Algorithms

Greedy Analysis Strategies:

- *Greedy algorithm stays ahead.* Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.
- *Exchange argument.* Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

① Interval Scheduling

Consider jobs in increasing order of finish time.

② Scheduling to Minimize Lateness

Earliest deadline first.

③ Optimal Caching

Farthest-in-future. Evict item in the cache that is not requested until farthest in the future.

④ Clustering

Single-link k-clustering algorithm.

L04 Divide and Conquer

设 $a \geq 1$ 和 $b > 1$ 为常数，设 f(n)为一函数，T(n)由递归式

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

其中$\frac{n}{b}$指$\lfloor \frac{n}{b} \rfloor$和$\lceil \frac{n}{b} \rceil$，可以证明，略去上下去整不会对结果造成影响。那么 T(n)可能有如下的渐进界

(1)若 $f(n) < n^{\log_b a}$，且是多项式的小于。即

$\exists\, \varepsilon > 0$，有 $f(n) = O(n^{\log_b a - \varepsilon})$，则 $T(n) = \Theta(n^{\log_b a})$

(2)若 $f(n) = n^{\log_b a}$，则 $T(n) = \Theta(n^{\log_b a} \log n)$

(3)若 $f(n) > n^{\log_b a}$，且是多项式的大于。即

$\exists\, \varepsilon > 0$，有 $f(n) = \Omega(n^{\log_b a + \varepsilon})$，且对$\forall\, c < 1$ 与所有足够大的 n，有 $af\left(\frac{n}{b}\right) \leq cf(n)$，则 $T(n) = \Theta(f(n))$

① Merge sort

Def. T(n) = number of comparisons to mergesort an input of size n.

$$T(n) = \begin{cases} 0 & if\ n = 1 \\ T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right) + n & otherwise \end{cases}$$

$T(n) = O(n \log_2 n)$.

② Closest Pair of Points

$T(n) \leq 2T(n/2) + O(n) \rightarrow T(n) = O(n \log n)$

③ Integer Multiplication

Divide each n-digit integer into two ½n-digit integers (Karatsuba-Ofman, 1962)

$$xy = 2^n \cdot x_1 y_1 + 2^{\frac{n}{2}} \cdot ((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0) + x_0 y_0$$

$$T(n) \leq T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + T\left(1 + \lceil \frac{n}{2} \rceil\right) + \Theta(n)$$

$$T(n) = O(n^{\log_2 3}) = O(n^{1.585})$$

④ Matrix Multiplication

Divide each n-by-n matrix into four ½n-by-½n blocks

7 multiplications, 18 additions.

$T(n) \leq 7T\left(\frac{n}{2}\right) + \Theta(n^2) \rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$

Best known. $O(n^{2.3728596})$ [Alman & Williams, 2020]

Conjecture. $O(n^{2+\epsilon})$ for any $\epsilon > 0$.

⑤ Convolution and FFT

$$A(x) = A_{even}(x^2) + xA_{odd}(x^2).$$

$$A(-x) = A_{even}(x^2) - xA_{odd}(x^2).$$

Combine.

$$A(\omega^k) = A_{even}(v^k) + \omega^k A_{odd}(v^k), 0 \leq k < n/2$$

$$A(\omega^{k+n/2}) = A_{even}(v^k) - \omega^k A_{odd}(v^k), 0 \leq k < n/2$$

Integer multiplication

Convert to binary polynomial, then multiply.

O(n log n) complex arithmetic steps.

L05 Dynamic Programming

Top-down: May skip unnecessary sub-problems

Bottom-up: Save the overhead in recursion

① Weighted Interval Scheduling - $O(n \log n)$

[O(n) if pre-sorted start & finish]

OPT(j) = value of optimal solution to the problem consisting of job requests 1, 2, ..., j.

$$OPT(j) = \begin{cases} 0 & if\ j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & otherwise \end{cases}$$

② Knapsack Problem - $\Theta(nW)$

Def. OPT(i, w) = max profit subset of items 1, …, i with weight limit w.

$$OPT(i,w) = \begin{cases} 0 & if\ i = 0 \\ OPT(i-1,w) & if\ w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1, w-w_i)\} & otherwise \end{cases}$$

③ RNA Secondary Structure – $O(n^3)$

OPT(i, j) = maximum number of base pairs in a secondary structure of the substring $b_i b_{i+1} \ldots b_j$.

$$OPT(i,j) = \begin{cases} 0 & i \geq j-4 \\ \max(OPT(i,j-1), 1 + \max_t\{OPT(i,t-1) + OPT(t+1,j-1)\}) & i < j-4 \end{cases}$$

④ Sequence Alignment - $\Theta(mn)$time and space

Edit distance. Gap penalty δ; mismatch penalty α_{pq}.

Def. OPT(i, j) = min cost of aligning strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.

$$\text{OPT(i, j)} = \begin{cases} j\delta & if\ i = 0 \\ min\begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & otherwise \\ i\delta & if\ j = 0 \end{cases}$$

⑤ Sequence Alignment in Linear Space

-$O(mn)$ time to compute $f(\bullet, n/2)$ and $g(\bullet, n/2)$ and find index q.

- $T(q, n/2) + T(m-q, n/2)$ time for two recursive calls.

$T(m, n) \leq cmn + T(q, n/2) + T(m-q, n/2)$

⑥ Shortest Paths

Def. OPT(i, v) = length of shortest v-t path P using at most i edges.

$$\text{OPT(i,v)} = \begin{cases} \infty & if\ i = 0, v \neq t \\ 0 & if\ v = t \\ \min\left\{OPT(i-1, v),\ \min_{(v,w) \in E}\{OPT(i-1, w) + c_{vw}\}\right\} & otherwise \end{cases}$$

if no negative cycles, then OPT(n-1, v) = length of shortest v-t path.

$\Theta(mn)$ time, $\Theta(n^2)$ space.

O(n) extra space, $O(mn)$ time

⑦ Distance Vector Protocol

Bellman-Ford "Routing by rumor." each router performs n separate computations, one for each potential destination node. "counting to infinity" Each router also stores the entire path. Requires significantly more storage.

⑧ Negative Cycles in a Graph

Can detect negative cost cycle in O(mn) time. Add new node t and connect all nodes to t with 0-cost edge. Check if OPT(n, v) = OPT(n-1, v) for all nodes v.

- if yes, then no negative cycles
- if no, then extract cycle from shortest path from v to t.

L06 Network Flow

① Residual Graph, Augmenting Path, Ford-Fulkerson Algorithm

Def. The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e\ out\ of\ A} c(e)$.

Capacity Scaling: $\mathbf{O(m^2 \log C)}$ time.

② Bipartite Matching

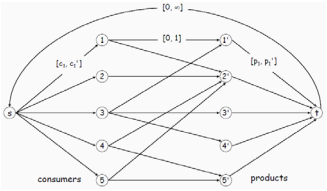
Perfect Matching: |L| = |R|, G has a perfect matching iff $|N(S)| \geq |S|$for all subsets $S \subseteq L$.

③ Extensions to Max Flow

Circulation with Demands. Add new source s and sink t. For each v with d(v) < 0, add edge (s, v) with capacity -d(v). For each v with d(v) > 0, add edge (v, t) with capacity d(v).

④ Survey Design

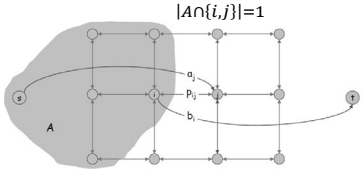
Integer circulation = feasible survey design.



⑤ Image Segmentation

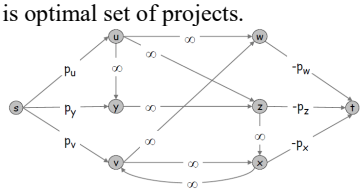
Find partition (A, B) that maximizes:

$\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E,\ |A \cap \{i,j\}|=1} p_{ij}$



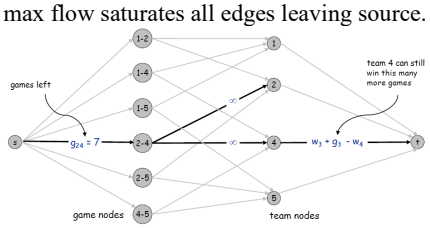
⑥ Project Selection

Min cut formulation. (A, B) is min cut iff A − {s} is optimal set of projects.



⑦ Baseball Elimination

Assume team 3 wins all remaining games. $w_3 + g_3$ wins. Team 3 is not eliminated iff max flow saturates all edges leaving source.



Explanation for Sports Writers

Team z is eliminated iff there exists a subset T* such that $\frac{w(T^*)+g(T^*)}{|T^*|} > w_z + g_z$.

Define T* = team nodes on source side of min cut. Observe x − y ∈ A iff both x ∈ T* and y ∈ T*. $g(S - \{z\}) > cap(A, B)$.

L07 NP and Computational Intractability

① Polynomial-Time Reductions

Problem X polynomial-time *reduces to* problem Y if arbitrary instances of problem X can be solved using polynomial number of standard computational steps, plus polynomial number of calls to oracle that solves problem Y. (not reduce from)

② NP stands for nondeterministic polynomial-time.

P. Decision problems for which there is a poly-time algorithm.

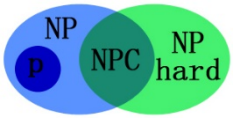
EXP. Decision problems for which there is an exponential-time algorithm.

NP. Decision problems for which there is a poly-time certifier.

③ Terminology

NP-complete. A problem in NP such that every problem in NP polynomial reduces to it. **NP-hard.** A problem such that every problem in NP reduces to it.

co-NP.



Complements of decision problems in NP. for *no* instance, there is a succinct disqualifier. (e.g. TAUTOLOGY, NO-HAM-CYCLE, PRIMES.)

- If NP≠co-NP, then P≠NP.

- P ⊆ NP ∩ co-NP.
- Factoring is in NP∩co-NP, but not known to be in P. (Factor: Given two integers x and y, does x have a nontrivial factor less than y?)

③ NP-complete.
Step 1. Show that Y is in NP. (polytime cert)
Step 2. Choose an NP-complete problem X.
Step 3. Prove that $X \leq_p Y$.

CIRCUIT-SAT. Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?
3-SAT. Given CNF formula Φ, each clause contains exactly 3 literals, does it have a satisfying truth assignment?
INDEPENDENT SET. Given a graph G = (V, E) and an integer k, is there a subset of verticesS ⊆ Vsuch that |S| ≥ k, and for each edge at most one of its endpoints is in S?
VERTEX COVER. Given a graph G = (V, E) and an integer k, is there a subset of verticesS ⊆ Vsuch that |S| ≥ k, and for each edge, at least one of its endpoints is in S?
SET COVER. Given a set U of elements, a collection $S_1, S_2, \dots, S_m$ of subsets of U, and an integer k, does there exist a collection of ≤ k of these sets whose union is equal to U?
HAM-CYCLE. given an undirected graph G = (V, E), does there exist a simple cycle Γ that contains every node in V.
DIR-HAM-CYCLE. given a digraph G = (V, E), does there exists a simple directed cycle Γ that contains every node in V?
TSP. Given a set of n cities and a pairwise distance function d(u, v), is there a tour of length ≤ D?
LONGEST-PATH. Given a digraph G = (V, E), does there exists a simple path of length at least k edges?
3D-MATCHING. Given disjoint sets X, Y, and Z, each of size n and a set T ⊆ X × Y × Z of 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-COLOR. Given an undirected graph G does there exist a way to color the nodes red, green, and blue so that no adjacent nodes have the same color?
k-REGISTER-ALLOCATION. Assign program variables to machine register so that no more than k registers are used and no two program variables that are needed at the same time are assigned to the same register.
SUBSET-SUM. Given natural numbers $w_1, \dots, w_n$ and an integer W, is there a subset that adds up to exactly W?
PARTITION. Given natural numbers $v_1, \dots, v_m$, can they be partitioned into two subsets that add up to the same value?
SCHEDULE-RELEASE-TIMES. Given a set of n jobs with processing time t_i, release time r_i, and deadline d_i, is it possible to schedule all jobs on a single machine such that job i is processed with a contiguous slot of t_i time units in the interval $[r_i, d_i]$?

L08 PSPACE

① Quantified satisfiability: Let $\Phi(x_1, \dots, x_n)$ be a boolean CNF formula. Is the following propositional formula true?
$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots \forall x_{n-1} \exists x_n \ \Phi(x_1, \dots, x_n)$
Q-SAT ∈ PSPACE-complete
PLANNING ∈ PSPACE
Competitive facility location ∈ PSPACE-complete
Input. Graph G = (V, E) with positive edge weights, and target B.
Game. Two competing players alternate in selecting nodes. Not allowed to
select a node if any of its neighbors has been selected.
Can second player guarantee at least B units of profit?

L09 Extending tractability

① Vertex Cover: small k – pick an edge, remove any vertex of the two, # of edges ≤ k(n − 1). $O(2^k kn)$
② Independent set on trees: greedy pick leaf $O(n)$; Weighted: DP. w/,w/o root, $O(n)$.
③ Circular arc coloring: DP, $O(k!\,n)$. List all possibilities. For small k.
④ Vertex cover in bipartite graphs: max matching = min vertex cover. Network flow.

L10 Local Search

① Gradient descent: vertex cover, remove dots.
② Metropolis algorithm: update with prob $e^{-\Delta E/(kT)}$ $(\Delta E > 0)$. Simulated annealing.
③ Hopfield neural networks: flip bad nodes. Progress $\Phi(S) = \sum_{e\ good} |w_e|$. $W = \sum_e |w_e|$.
④ Maximum cut: Single-flip neighborhood, greedy. Big-improvement-flip: flip increase $\geq \frac{2\epsilon}{n} w(A, B)$ => $(2 + \epsilon)w(A, B) \geq w(A^*, B^*)$. $O(\epsilon^{-1} n \log \sum_e w_e)$ flips; KL-neighborhood.
⑤ Nash equilibria.

L11 Lower Bounds

① If an algorithm takes too little time, it must sometimes produce the wrong answer.
② Merge two list: 2n-1; Max: n-1; Max-min: 3n/2-2; Sort: $\Omega(n \log n)$

L12-15 Randomized algorithms

① Max-cut, Monte Carlo, random put, in expectation 2-approximation.
② Quicksort, random pivot.
③ Hash table. Closed/open addressing. Load factor α. Universal hashing: $\Pr_{h \in H}[h(x) = h(y)] = 1/m$. Perfect hashing.
④ Bloom filters: $\left(1 - \frac{1}{m}\right)^{nk} \approx e^{-\frac{nk}{m}}$.
⑤ Fingerprint. $O(\log n)$bits, $O(1/n)$ FP.
⑥ String matching. Monte Carlo, O(n+m), FPprob O(1/n). Las Vegas, expected O(m+n).
⑦ **Union Bound.** If $Pr[E_i] = p_i$ $(1 \leq i \leq k)$, then $\Pr[E_1 \cup \dots \cup E_k] \leq p_1 + \dots + p_n$.
⑧ **Markov's Inequality.** Positive r.v. X: $\Pr[X \geq a] \leq E[X]/a$ $(a > 0)$.
⑨ **Chebyshev's Inequality.** $\Pr[|X - E[X]| \geq a] \leq Var[X]/a^2$ $(a > 0)$.
⑩ **Chernoff Bounds.**
(1) Let $X_1, X_2, \dots, X_n$ be independent r.v.s, with values in {0,1} s.t. $E[X_i] = p_i$ for all i. Let $X = \sum_i X_i$ and $\mu = E[X] = \sum_i p_i$. Then
For $0 < \delta \leq 1, \Pr[X \geq (1 + \delta)\mu] \leq e^{-\mu\delta^2/3}$.
For $\delta > 1, \Pr[X \geq (1 + \delta)\mu] \leq e^{-\mu\delta \ln \delta/3}$.
For $0 \leq \delta \leq 1, \Pr[X \leq (1 - \delta)\mu] \leq e^{-\mu\delta^2/2}$.
(2) Same setting as above. For any $\delta > 0$,

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

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

(3) Let $X_1, X_2, \dots, X_n$ be independent {-1, 1} valued r.v.s, with $Pr[X_i = 1] = Pr[X_i = -1] = 1/2$ for all i. Let $X = \sum_i X_i$. Then for any $\delta \geq 0$,

$$Pr[X \geq \delta] = Pr[X \leq -\delta] \leq e^{-\frac{\delta^2}{2n}}$$

(4) Two-sided Chernoff Bound
Let $X_1, X_2, \dots, X_n$ be independent r.v.s, $0 \leq X_i \leq 1$. Let $X = \sum_i X_i$ and $\mu = E[X] = \sum_i E[X_i]$. Then for any $\epsilon > 0$,

$$Pr[|X - \mu| \geq \epsilon\mu] \leq 2\exp(-\frac{\epsilon^2}{2 + \epsilon}\mu)$$

$(2 + \epsilon \ can\ be\ replaced\ with\ 3.)$

More applications: Load balancing ($m = 16n \ln n$), Set balancing ($\sqrt{4m \ln n}$?), 2D LP (O(n)), d-D LP (O(d!n)) …

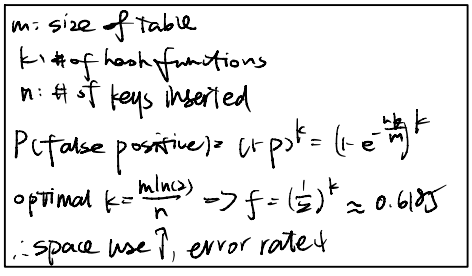
L15-17 Approximation algorithms

① Set covering. $\ln(n)$-approximation. Pick the cheapest.
② Makespan scheduling. NPC. List scheduling: 2-approximation. Longest processing time (LPT) schedule: 4/3-approximation.
③ The knapsack problem. polynomial time approximation scheme (PTAS), $(1 + \epsilon)$ -approximation. Run time $O(n^3/\epsilon)$. Scaling factor $\theta = \epsilon v^*/2n$.
④ Vertex cover: 2-approximation. Weighted vertex cover: Integer linear programming

$$\begin{aligned} (ILP)\quad \min\ & \sum_{i \in V} w_i x_i \\ \text{s.t.}\ & x_i + x_j \geq 1 \qquad (i,j) \in E \\ & x_i \in \{0, 1\} \qquad i \in V \end{aligned}$$

$$\begin{aligned} (LP)\quad \min\ & \sum_{i \in V} w_i x_i \\ \text{s.t.}\ & x_i + x_j \geq 1 \qquad (i,j) \in E \\ & x_i \geq 0 \qquad i \in V \end{aligned}$$

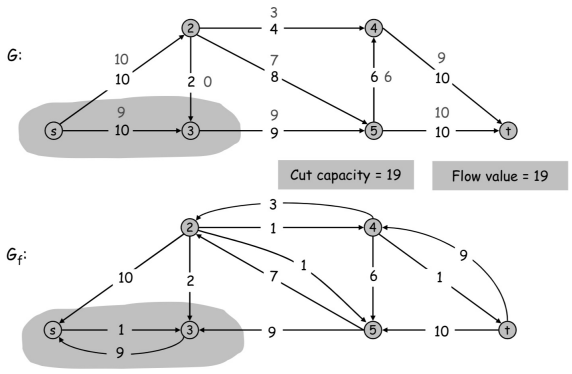
$S = \{i \in V : x_i^* \geq 1/2\}$. 2-approximation.
⑤ Traveling Salesman Problem (TSP)
Metric TSP (triangle inequality):
2-approximation, minimum spanning tree + DFS; 1.5-approximation, MST + odd degree perfect matching + Euler path + short cut.
⑥ k-Center problem (triangle inequality)
NPC, Gonzalez's algorithm, 2-approximation. Add center to farthest site.



Each node tries to send w.p $\frac{1}{n}$
Pr[i fails to send in t-th round] ≤ $1 - \frac{1}{en}$
After $2en \ln(n)$ rounds, all nodes succeed sending w.p ≥ $1 - \frac{1}{n}$

Maximal independent Set: independ vertex
≠ local optimal ≠ MaxIS: global max

? Each node v chooses a random number r(v) ∈ [0,1] and sends it to its neighbors.
? If r(v)<r(w) for all neighbors w of v, then v adds itself to the MIS and informs its neighbors.
? If v or one of its neighbors entered the MIS, v terminates. Remove all of v's edges.
? Otherwise go back to first step, until graph is empty.



2D LP algorithm

- If |H|=2, output intersection of the 2 halfplanes.
- Pick random constraint h∈H.
- Recursively find opt=B(H-{h}).
- If opt doesn't violate h, output opt.
  - opt violates h if opt lies outside h.
- Else project H-{h} onto h's boundary to obtain a 1D LP.
- Output the opt of the 1D LP.

$T(n) \le T(n-1)+O(1)+2/n(O(n)+O(1))$. $\Rightarrow T(n) = O(n)$

d-Dimensional LP algorithm

- If |H|=d, output their intersection.
- Pick random constraint h∈H.
- Recursively find opt=B(H-{h}).
- If opt doesn't violate h, output opt.
- Else project H-{h} onto h's boundary to obtain a d-1 dimensional LP.
- Recursively solve the d-1 dim LP.

$T(n,d) \le T(n-1,d)+O(d)+d/n(O(dn)+T(n-1,d-1))$. $\Rightarrow T(n) = O(d!\,n)$

Set covering

- U = X
- C = ∅
- while U≠∅
  - choose S∈F-C with min |cost(S)|/|S∩U|
  - C = C∪{S}
  - U = U – S
- output C

Per unit cost to cover new elements.

- Order the sets in C by when they're added to C, earliest set first.
  - Let the order be $S_1, S_2, \ldots, S_m$.
- Cost of the set cover is $L=\Sigma_i \text{cost}(S_i)$.
- Order the elements in X by when they're added, earliest element first.
  - Let the order be $e_1, e_2, \ldots, e_n$.
  - So, the first few e's are added by S_1, the next few added by S_2, etc.
  - Every element in X is in the list, because C covers X.
- Let n_i be the number of new elements S_i covers.
  - So, n_i is the number of elements in S_i, but not in $S_1, \ldots, S_{i-1}$.
- Divide the cost of S_i evenly among the new elements it covers.
  - If e is newly covered by S_i, then $\text{cost}(e) = \text{cost}(S_i)/n_i$.
- $\Sigma_k \text{cost}(e_k) = \Sigma_i n_i \cdot \text{cost}(S_i)/n_i = \Sigma_i \text{cost}(S_i) = L$.
  - Every element is covered by some S_i, and S_i covers n_i new elements.
- We'll prove $\text{cost}(e_k) \le \text{OPT}/(n-k+1)$, for any k.
- Suppose this is true, then

$L = \Sigma_k \text{cost}(e_k) \le \Sigma_k \text{OPT}/(n-k+1) \approx \ln(n)\cdot\text{OPT}$

vertex cover

make |V'| as small as possible

- Initially, let D be all the edges in the graph, and C be the empty set.
  - C is our eventual vertex cover.
- Repeat as long as there are edge left in D.
  - Take any edge (u,v) in D.
  - Add {u,v} to C.
  - Remove all the edges adjacent to u or v from D.
- Output C as the vertex cover.

2-approximation

? Let C* be an optimal vertex cover.
? Let A be the set of edges the algorithm picked.

? None of the edges in A touch each other.
? Each time we pick an edge, we remove all adjacent edges.
? So each vertex in C* covers at most one edge in A. ? The 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 number of vertices the algorithm uses is 2|A|.
? If alg picks edge (u,v), it uses {u,v} in the cover.
? So (# vertices alg uses) / (# vertices in opt cover) = 2|A| / |C*| ? 2|A| / |A| = 2.

Makespan scheduling — NPC

- SUBSET-SUM problem: Given a set of numbers S and target t, is there a subset of S summing to t?
  - Ex S={1,3,8,9}. For t=9, yes. For t=14, no.
  - SUBSET-SUM is NP-complete. Will reduce it to 2 processor makespan scheduling.
- Let (S,t) be an instance of SUBSET-SUM, and let s be sum of all elements in S.
- Make a set of tasks J = S∪{s-2t}, and schedule them on 2 processors.

Claim If some subset of S sums to t, then min makespan is s-t

? If there are n tasks and m processors, list scheduling only takes O(n log m) time.

2-approximation

? Suppose LS gives makespan of M.
? Let the optimal schedule have makespan M*.

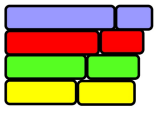
? Consider task X that finishes last.
? Say X starts at time T, and has length t.
? Claim1M* ge t.
Claim2 M* ge T.

- So M* ≥ max(T,t).
- M = T + t, because X is last job to finish.
- So M/M* ≤ (T+t)/max(T,t) ≤ 2.

LPT scheduling: 先sort 再LS

$\frac{4}{3}$ – approx

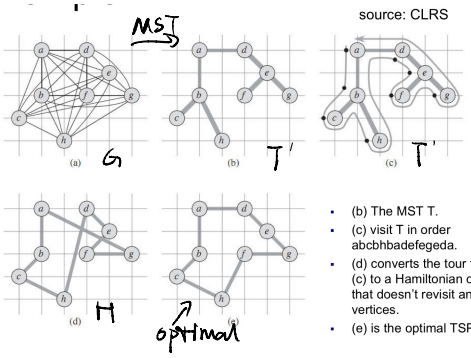
- Claim 1 LPT's makespan = T+t ≤ M*+t.
  - As in LS, no processor is idle up to time T, so M* ≥ T.
- Case 1 t ≤ M*/3.
  - Then LPT's makespan ≤ M* + t ≤ M* + M*/3 = 4/3 M*.
- Case 2 t > M*/3.
  - Since X is the smallest task, all tasks have size > M*/3.
  - So the optimal schedule has at most 2 tasks per processor. So n ≤ 2m.
  - If 1 ≤ n ≤ m, then LPT and optimal schedule both put one task per processor.
  - If m < n ≤ 2m, then optimal schedule is to put tasks in nonincreasing order on processors 1,...,m, then on m,...,1.
    - LPT also schedules tasks this way, so it's optimal.



Travelling Salesman

a circle that visit each node once.
滿足三角不等式的，有1.5-approx TSP

source: CLRS



- (b) The MST T.
- (c) visit T in order abcbhbadefegeda.
- (d) converts the tour from (c) to a Hamiltonian cycle, that doesn't revisit any vertices.
- (e) is the optimal TSP.

Lemma If H is the shortcut of T', then c(H) <=c(T').

2-approx

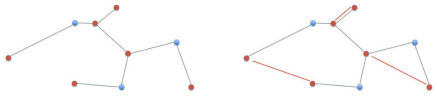
- Let H* be an optimum TSP.
- If we delete an edge from H*, we get a spanning tree.
- Since T is an MST, c(T) ≤ c(H*).
- Call the path from the depth-first traversal T'.
  - T' crosses each edge in T twice.
  - So c(T') = 2 c(T).
- Let H be the outcome of shortcutting T'.
  - H is a Hamiltonian cycle. It visits all the vertices, ar ends where it started.
  - c(H) ≤ c(T'), by the lemma.
  - c(H) ≤ c(T') = 2 c(T) ≤ 2 c(H*).
- So H is a 2-approximation.

An Euler tour of a graph is a path that starts and ends at the same vertex, and visits every edge once.

Thm (Euler) A graph has an Euler tour if and only if all vertices have even degree.,

Christofides 3/2-approx algorithm

- A 3/2-approximation for TSP with triangle inequality.
- Construct a minimum spanning tree T on G.
- Find the set V' of odd degree vertices in T.
- Construct a minimum cost perfect matching M on V'.
- Add M to T to obtain T'.
- Find an Euler tour T'' in T'.
- Shortcut T'' to obtain a Hamiltonian cycle H. Output as the TSP.



Proof of correctness

Lemma T' has an Euler tour.

Lemma Let H* be an optimal TSP on G, and let m be the cost of M. Then m ≤ c(H*)/2.

Proof Let H' be the optimal TSP on V'.
- c(H') ≤ c(H*) because H' is an optimal TSP on fewer vertices.
- H' is a cycle on V', so it consists of two matchings on V'. The cheaper one has cost m' ≤ c(H')/2 ≤ c(H*)/2.
- m ≤ m' because M has min cost.

- Thm Let H be the TSP output by Christofides and let H* be an optimal TSP. Then c(H) ≤ 3/2*c(H*).
- Proof
  - c(T) ≤ c(H*) because T is an MST.
  - c(T') = c(M) + c(T) ≤ c(H*)/2 + c(H*) = 3/2*c(H*).
  - c(H) ≤ c(T') because H is the shortcut of T'.

If $T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^d)$ for constants $a > 0, b > 1, d \geq 0$, then:

$$T(n) = \begin{cases} \Theta(n^d) & if\ d > log_b a \\ \Theta(n^d log n) & if\ d = log_b a \\ \Theta(n^{log_b a}) & if\ d < log_b a \end{cases}$$

Network Flow

duality between ***max flow*** and ***min cut***

The weight of the edge stands for its capacity.

Flows

s-t flow

- Each edge can have an amount of flow less than its capacity.
  - $0 \leq f(e) \leq c(e)$ for each $e \in E$
- The sum of in-flow equals the sum of out-flow (except the source and sink)
  - $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ for each $v \in C - \{s, t\}$

value of a flow f

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

Greedy

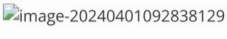
- 单走一条source 到 t 的路径，取上面所有edge的剩余的capacity的最小值作为f(e)
- 重复此步骤直到stuck

not always optimal

- 局部最优无法达到全局最优

undo to go out of the trapped decision

Residual Graph



Augmenting Path

a simple $s - t$ path P in the residual graph G_f.

Ford-Fulkerson Algorithm

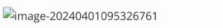
可以反悔的greedy

如果没有e^R这条反向edge每次循环后的添加，就等于纯粹的greedy

Cuts

An ***s-t cut*** is a partition (A, B) of V with $s \in A$ and $t \in B$.

The ***capacity*** of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of A}} c(e)$



Flow value lemma

Let v be any vlow, and let (A, B) be any s-t cut.
Then, the net vlow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of A}} f(e) - \sum_{e \text{ in to A}} f(e) = v(f)$$

Weak Duality

Let v be any flow, and let (A, B) be any s-t cut. Then thenvalue of the flow is at most the capacity of the cut.

Corollary. Let v be any vlow, and let (A, B) be any cut.
Iv v(v) = cap(A, B), then v is a max vlow and (A, B) is a min cut.

the equivalence of the following three conditions for any flow f:

- There exists a cut (A, B) such that v(v) = cap(A, B).
- Flow v is a max vlow.
- There is no augmenting path relative to v.

Choosing Good Augmenting Paths

Assumption. All capacities are integers between 1 and C.

n 是所有edge capacity之和？

Running time of Ford-Fulkerson is $O(mnC)$.

Load balancing

- 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 $\mathbf{Pr}[\max_i X_i > 2\mu] \leq n\frac{1}{n^2} = \frac{1}{n}$.
- By part 3 of Theorem 1, $\mathbf{Pr}[X_i < \frac{1}{2}\mu] \leq e^{-(\frac{1}{2})^2 16 \ln n/2} = e^{-2 \ln n} = \frac{1}{n^2}$.
- So by the union bound, $\mathbf{Pr}[\min_i X_i < \frac{1}{2}\mu] \leq \frac{1}{n}$.
- 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 r.v.'s, we can apply Thm 3 to bound $\max_i B_i$.

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

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

- We showed the probability item i's imbalance is $\geq \sqrt{4m \ln n}$ is $\leq \frac{1}{n^2}$. By the union bound, the probability that any of the n items' imbalance is $\geq \sqrt{4m \ln n}$ is $\leq \frac{1}{n}$. So w.h.p., the max imbalance is $\leq \sqrt{4m \ln n}$.