CO353 course note

Chenxuan Wei

Jan 2022

Contents

1 Week 1

1.1 Graph preliminaires

1. A graph is a tuple $(V, E \subseteq (v, v))$
   - uv-path is a sequence of vertices from u to v
   - cycle is a sequence of vertices where start and end are the same
   - connected if $\forall u, v \in V$, there is a uv path
   - cryclic = no cycle in the graph
   - a tree is a connected cryclic graph
2. Spanning Tree
   Let $G = (V, E)$ be a connected graph, and $T = (V_T, E_T)$
   - $E_T \subseteq E$
   - $V_T = V$

   then it is a spanning tree
3. Minimal spanning tree
   Assume C is a cycle, then $C - \{e\}$ is still a connected graph
   So if G is a connected graph and contains a cycle C, then $G - \{e\}$ is still a connected graph
   Hence $G - \{e\}$ is a minimal connected subgraph of G.
4. Directed graph: each edge has a direction and goes from a node to a node
   i.e. $(1, 2) \in E$ but $(2, 1) \notin E$

1.2 Shortest Paths: Dijkstra's algorithm

1. Shortest path problem
   Given directed graph $G = (V, E)$ with edge cost $\{c_e \geq 0\, e \in E\}$, find st path with min cost
   with $C(P) = \sum_{e \in P} c_e$ denote the cost
   let $d(u) = min(c(p))$ be the shortest path cost from s to u

2. Some idea
   If $(u, v) \in E$ then we have $d(v) \leq d(u) + C(u, v)$

3. Dijkstra's Algorithm
   Let G = (V, E) be directed graph, cost $c_e, e \in E$ find min sv path cost

   - Initial set $A = \{s\}, d(s) = 0, \forall v \neq s, l(v) = \infty$
   - While $A \neq V$
     - $\forall v \notin A$ such that $\exists u \in A$ with $(u, v) \in E$, update $l(v) = min\{l(v), (d(u) + C_{(u,v)})\}u \in A, (u, v) \in E$
     - select $w \in V - A$ such that l(w) has minimum $l(v), v \notin A$
     - update $A = A \cup \{w\}$, d(u) = l(u)

4. Remark

   - Can get the shortest path by a get the node $u \in A$ that determins l(w)
   - the path we get in first remark is a directed tree

1.3 Shortest Paths: correctness

1. Theorem 1.1
   If w is added to A in step 2(c), then $d^{alg}(w) = d(w)$

1.4 Running time and efficient algorithms

1. Runtime
   Number of elementary operation performed by algorithm as a function of input size

   - elementary operation
     - Addition, subtration, multiplication, division
     - comparision
     - simple logical: if, else
     - Assignments
   - input size
     # of bits needed to specify the input
     bits for integer = $log_2(x)$

2. Reasonable Running time
   is a polynimial of input size =¿ big O notation

1.5 Big-O notation

1. Definition
   Given function $f : R^n \rightarrow R$ and $g : R^n \rightarrow R$, we say
   $f(n) = O(g(n))$ if $\exists$ consts $c > 0$ and $n_0 \geq 0$ s.t. $f(n) \leq c * g(n), \forall n \geq n_0$

2. More definition
   $\exists c$ s.t. $f(n) = O(n^c), \forall n \geq n_0$

3. efficient
   $= f(n) = O(n^{O(1)})$

1.6 Dijkstra's algorithm

1. first idea

   - let $m = |E|, n = |V|$
   - there are n iterations of while loop
   - In each iteration:
     - computing l(v) takes $O(d^{in}(v))$, $d^{in}(v)$ = number of edges entering v (step 1)
     - computiing $l(v), \forall v$ takes O(m) times since we gothrough each edges once (step 2)
     - take O(n) times to compare min l(v) value for all nodes max n nodes → O(n) run time
     - takes O(1) to assign d(w) = l(w)

     So each interation takes $O(m+n) = O(m)$ since we need a connected graph which means $m \geq n$
   - Overall running time = O(mn) → poynomial of input → efficient

2. Better Implemetation
   if $\{u \in A : (u, v) \in E\}$ does not change across interations, then $l(v)$ does not change
   So we don't recomputing all $l(v), \forall v \notin A$, we do the following:

   - when pick $w \notin A$ to add to A, we only update $l(v), \forall v \notin A$ s.t. $(w, v) \in E$, and set
     $l^{new}(v) = min(l^{odd}(v), d(w) + C_{w,v})$

   So we will change the while loop to be
   Let $w*$ = last node added to A, Initially $w* = s$

   - $\forall (w*, v), v \notin A$ update $l(v) = min(l(v), d(w*) + C_{(w*,v)})$ Decrease key operation
   - Find $w \notin A$ with min $l(w)$ value Extract min operation
   - $A = A \cup \{w\}$, d(w) = l(w), w*=w

   So we examine each edge(u,v) at most once in step (a)

   (a) Using a simple array to store l() values:
      - DecKey: O(1)
      - Extract Min: O(n) times

      Runtime $= O(m + n^2) = O(n^2)$ (go throught each edges once only, then n nodes for each operation, for n interation)

   (b) Priority Queue
      - DecKey: O(log(n))
      - Extract Min: O(log(n)) times

      Runtime= $O(mlogn)$

   (c) Fibonaci heap
      - DecKey: O(1)
      - Extract Min: O(log(n)) times

      Runtime= $O(m + n * logn)$

2 Week 2

2.1 Minimum spanning trees

1. Spanning tree
   Let G = (V, E)
   A spanning tree $T = (V_T, E_T)$ satisified $E_T \subseteq E$, $V_T = V$
   and this will be a connected subgraph with minimum edges.

2. Minimum Spanning Tree problem
   Given a connected undirected graph $G = (V, E)$ edge cost $\{c_e\}, e \in E$, with no restirction on any c_e
   find a spanning tree with minimum total edge cost
   Since V doesn't change, we will just consider edges now
   use $C(T) = \sum_{e \in T} c_e$

3. Note

   - $c_e \in R$
   - if $c_e \geq 0, \forall e \in E$, then it is equvilenty as MST problem:find the min-cost connected subgraph of G
     Since there is always a optimal solution that is minimal connected subgraph

4. Fundemental theorem of tree (2.1)
   Let T = (V, F) let $n = |v|$, then if one of following is true, all is true

   - T is a tree
   - T is a connected graph has n-1 edges
   - T is a acylic, has n-1 edges

2.2 Cut property

1. Notations
   $\delta(v)$ whrere $v \in V$ is the set of edges incident to v
   $\delta(s)$ where $s \subseteq V = \{uv \in E : u \in S, v \not inS\}$

2. Assume: All edge costs are unique

3. Cut
   is a partition (A, V - A) of the vertex set V, where $A \neq \emptyset$, $A \subset V$
   $\delta(A) = \delta(V - A)$ which is the edges divide the vertices
   if $F \in E$ and $F \cap \delta(A) \neq \emptyset$ then F is in the cut

4. Lemma 2.2: Cut property
   Consider any cut (A, B), where $B = V - A$
   If e is the unique min-cost edges across the cut, then e belong to every MST

2.3 Prim's Algorithm

1. Prim's Algorithm
   (a) Pick a seed node $s \in V$
   (b) Initilize $A = \{s\}$, $T = \emptyset$
   (c) while $A \neq V$
      - Choose $e = uv \in \delta(A)$ with min cost, where $u \in A, v \notin A$
      - $A = A \cup \{v\}$, $T = T \cup \{e\}$

   (d) return T, which is the set of edge for the MST
2. Theorem 2.3
   Prim's algorim correctly compute the MST
3. Corollary
   T is a unique MST
   Howeer, if edge cost is not distinct:
   - Prim's algorithm still works, return one of the MST
   - There will be mutiple MST
4. Running time
   For each interation
   (a) For each edge wv where $v \notin A$, update $a(v) = min(a(v), C_{wv})$ Decrease Key
   (b) Find $w \in V - A$ with smallest $a(w)$ extract min
   (c) $A = A \cup \{v\}$, $T = T \cup \{e\}$

   So run time $= O(m + n^2)$ if using simple array
   O(m+nlogn) using Fibonacci Heap

2.4 Kruskal's algorithm

1. Steps

   (a) Sort the edges in increasing order of cost

   (b) set $T = \emptyset$

   (c) For each edge e in sorted order

   - if $T \cup \{e\}$ does not create a cycle: $T = T \cup \{e\}$

   (d) Return T

2. Theorem 2.4: it returns a unique MST, when all cost are distinct

3. Running Time:

   - Sorting m edges takes $O(mlog(m))$ times, by quicksort
   - check if $e = uv$ can be added, will bedone in $O(logn)$ times
   - So total time for one interation is $O(mlogn)$ times, since we need to check all edges
   - Hence total is $O(mlogm + mlogn) = O(mlogn)$ since $m \leq n^2$

2.5 Clustering application

1. General iDea:
   Given a set of object and some notion with similarty and dissimilarty between them, and

   - Object in same group are "similar"
   - boject in different groups are "dissimilar" to each other

2. Maximum Spacing clustering
   Given : a set $V = \{p_1 \dots p_n\}$ of objects/points, and pariwise distance $d(p_i, p_j) = d(p_j, p_i) \geq 0$
   Goal: Partition V into k cluster $C_1 \dots C_k$ (So $C_i \cap C_j = \emptyset$, $\cup_{i=1}^k C_i = V$)
   to max the min inter-cluster spacing = min distance between a pair of points in different clusters
   which is $Max(min\{d(p,q) : p \in C_i, q \in C_j, i \neq j\})$

3. Algorithm: single - linkage clustering: agglomerative algorithm

   - start with put every point in a separte cluster
   - Repeatally merge the 2 clusters with smallest inter cluster distance, until k cluster left

4. Analysis algorithm
   Consider a graph G with V and edge between every pair of $p, q \in V, p \neq q$ with cost $d(p,q)$
   merge 2 cluster = adding edge pq
   stop when there are k components
   similar to apply kruskal algorithm and find a MST
   but delete k-1 most costly edges in the MST

2.6 Clustering application: Correctness

3 Week 3

3.1 Arborescences

1. Definition = Directed spanning tree
   Let $G = (V, E)$ be a directed graph, and let $r \in V$ be the root node
   An aborescence rooted at r is a subgraph $T = (V, F)$ s.t.

   - there is a $r \to v$ path $\forall v \in V$
   - T is a spanning tree if we ignore the direction (undiretive property)

2. Lemma 3.1: useful alternate characterictic of aborecences
   $T = (V, F)$ is aborescence rooted at r $\iff$ T has no directed cycle and $v \neq r$ have exactly one incoming edge

3.2 Min cost Arborescences

1. The problem
   Let $G = (V, E)$ be a directed graph, and root node $r \in V$ and edge costs $c_e, e \in E$
   And we want to find a arborescences with min total edge cost
   Assume $\forall v \in V, \exists r \to v$ path

2. Two greddy strategies(inspired by MST)

   - Pick the cheapest edge entering a node v
   - Consider somecycle, delete the most costly edge of the cycle

3. Observations

   - if $(V, F*)$ is an A, then it is an MCA, where $F*$ =all edge enter a v with min cost
     If it is not a A, then by lemma 3.1, it must have a cycle C
   - Suppose for every node $v \neq r$, for each edge e entering v, we subtract a common amount p_v
     for c_e where the edge entering v
     Then for any arborescence T, the cost is differ by a constant = $\sum_{v \neq r} P_v$

4. Corollary
   Let $y_v = min(C_{uv})$ and $c'_e = c_e - y_v \forall eenteringv, \forall v \neq r$
   Then T is an MCA with $\{c_e\}$ cost $\iff$ T is MCA with $\{c'_e\}$

3.3 Edmond's Algorithm

1. Algorithm
   Input:

   - Directed Graph $G = (V, E)$
   - root $r \in V$
   - $\{c_e\}$ edge cost
   - $\exists r \rightarrow V, \forall v \in V$

   (a) Base case: if only one node, return $\emptyset$

   (b) For each $v \neq r$
      - let $y_v = min(C_{uv}), uv \in E$
      - set $c'_e = c_e - y_v, \forall e entering v$

   (c) For each $v \neq r$, choose a 0 c' cost edge entering v, let $F*$ be the set of all edges

   (d) if $F*$ is A, return it

   (e) then it must contain a directed cycle, Let $Z \subseteq F*$ be the cycle, $r \notin Z$ Contract Z in to single super node, get graph $G' = (V', E')$

   (f) And recursily run find a MCA in G'

   (g) Extend (V', F') to an a (V, F) in G
      - Let $v \in Z$ be the node that has a incoming edge in F'
      - Let $F = F' \cup Z - \{$edge of z entering v$\}$

   (h) Return F

3.4 Analysis

1. Running time: Make O(m) operations and a recursive call to smaller graph, at most n recursive call
   make $O(mn)$ time

2. Lemma 3.2
   Suppose we have $\{d_e\}$ edge costs, and a 0 d-cost cycle Z
   Then $\exists$ an MCA with d-costs that has exatly one edge entering Z

4 Week 4

4.1 Matriods - definition and examples

1. Definition
   is a tuple M = (V, I), where U is a ground set, $I \subseteq 2^U$ is a collection of subset of U
   satisified the following property

   - $\emptyset \in I$
   - if $A \in I$ and $B \subseteq A$, then $B \in I$
   - exchange property, if $A, B \in I$ with $|A| < |B|$
     then $\exists e \in B - A$, s.t. $A \cup \{e\} \in I$

2. Terms

   - if $A \in I$, it is a independent set, if not, it is s dependent set
   - A maximal independent set, B
     s.t $B \cup \{e\} \notin I, \forall e \in U - B$, is called a basis
   - universial matriods
     $I = \{A \subseteq U, |A| \leq k\}$
   - partition matroid
     $I = \{A \subseteq U, |A \cap S_i| \leq r_i\}$
   - graphic matriod(cycle matriod)
     $I = \{A \subseteq E$ A is acylic $\}$

4.2 Matriod optimization

1. maximum weight indepdendent set problem (MWIS)
   Given M = (U, I), and $\{w_e\}, e \in U$
   find max-weight independent set, find $A \in I$, where $W(A) = \sum_{e \in A} w_e$ be maxed

2. Common independent set
   Given $M_1 = (U, I_1), M_2 = (U, I_2)$, and $\{w_e\}, e \in U$
   find maximum weight set is independent in both matroids

3. MWIS problem
   Sepcial case $\rightarrow$ MST problem

4. Greedy Algorithm for MWIS
   Define $M = (U, I)$, $\{w_e\}, e \in V$
   Assume $w_e > 0$
   Define $M' = (U' = \{e \in U, w_e \geq 0\}, I' = \{A \subseteq U' : A \in I\})$
   Algo:

   - sort element in U
   - Let $A = \emptyset$
   - considering element in sorted order
     if $A \cup \{e\}$ is indepdnent, then $A = A \cup \{e\}$
     if not, move to next one
   - return A

4.3 Corresness

Let A be the set returned by greedy

1. All basis in M have same length 4.1
   Proof: if not, then $|A| < |B|$, $A, B \in I$ trigger property c
   which shows $|A| \cup \{e\} \in I$ contradiction

2. A is a basis of M 4.2
   Proof: Assume it is not, then $\exists e \notin A$, $A \cup \{e\} \in I$
   then at some point in algorithm we should have $S \subseteq A$ where we are deciding if $\{e\}$ should be added
   Since $S \subseteq A$, we have $S \cup \{e\} \subseteq A \cup \{e\}$
   So $S \cup \{e\} \in I$ too by property b, so e can be added, but not right now, hence contradiction

3. Actual prove
   Let $A*$ be a MWIS, and this will be basis too
   hence by 4.1, we have $|A| = |A*|$
   Suppose w(A) < w(A*), with

   - $A = \{e_i\}$,
   - $A* = \{e*_i\}$
   - both weight in decreasing order

   And A_i, $A*_i$ be the first i element in them
   consider the smallest j s.t. $w(A_j) < w(A*_j)$
   and we know $|A_{j-1}| = j - 1 < j = |A*_j|$
   then by property c, we know $\exists e \in A*_j - A_{j-1}$, $A_{j-1} \cup \{e\} \in I$
   and claim $w_e \geq w_{e*_j} > w_{e_j}$

   - first is true since $e \in A*_j$, $e*_j$ will have least weight in $A*$
   - the second is true since j is smallest index difference appear
     so $w(A_{j-1}) = w(A*_{j-1})$, then we have the second is true

   Then when the algo runs on A_{j-1}, we know $A_{j-1} \cup \{e\} \in I$
   So we should add e instead of e_j hence contradiction

4.4 runtime

1. Independent orcicle
   a procedure that give a set $S \subseteq U$, return if $S \in I$

2. Running time

   - sorting takes $O(mlogm)$
   - O(1)
   - m calls to orcle

   So it is efficient

5 Week 5

5.1 Minimum Steiner tree

1. Definition
   Let $G = (V, E)$ be undirected $\{c_e\}, c_e \geq 0$, a set $T \subseteq V$ called terminal
   we want to find a MST for T, and this MST is called steiner tree T'
   for vertices in Steriner Tree but $\notin T$, is a steine node

2. Useful transformation: $(G, c, T) \rightarrow (G', c', T)$
   where $G' = (V, E')$ be complete graph on V
   and c'_{uv} = shortest path cost in G between u and v
   Note: $c'_{uw} \leq c'_{uv} + c'_{uw}$ (triangle inequaility)
   Let F' be MsinT in G', F be MsinT in G

3. Claims

   - Any steiner tree F in G also a steiner tree in G' and $c'(F) \leq c(F)$
   - Any steiner tree F' in G' yeilds a steiner tree F in G s.t. $c(F) \leq c'(F')$

4. Proofs for claims

   - It is clearly that since $E \subseteq E'$, also $\forall uv \in E$, $c'_{uv} \leq c'_{uv}$, so we get the result
   - Let $\overline{F} = \cup_{uv \in F'} P_{uv}$ where P_{uv} is the shortest uv-path in G
     Cleary $\overline{T}$ have all vertices in T, but cycle may appear, but it is easy to check/remove
     So $c(F) \leq c(\overline{F}) \leq \sum_{uv \in F'} c(P_{uv}) = \sum_{uv \in F'} c'_{uv} = c'(F')$

5. Metric completion
   (G', c') defined above is metric completion of (G, c)
   so we can always solve steiner tree problem on metric completion

5.2 algorithm

1. Algorithm:
   - Input: $G = (V, E), c_e \geq 0, T$
   - For $G'[T] = (T, E'[T])$, where for $uv \in E'[T], u, v \in T$, find a MST on it with c' costs
   - map it back to G by 5.1(b)
2. idea
   cost of solution ≤ MST(G', c', T)
   let OPT = OPT(G', c', T) = OPT(G, c, T), and they are optimal solution for the graph
3. Theorem 5.2: $F' = MST(G', c', T) \leq 2 * OPT$
4. Proof:
   let $F*$: optimal Ster T for (G', c', T)
   pick some $r \in T$, root F* at r, and do a DFS travelsal of $F*$ starting at r, the DFST return a tour that visit all node of $F*$, has c' cost $\leq 2c'(F*) = 2 * OPT$
   list out the node traveled for DFST, find the first occurance of $v \in T$
   and find cycle $\overline{Z}$ visist every terminal node exactly once
   $= r, v_1, v_2 \ldots v_{j-1}, r$ where v_I will not be visited
   then by triangle inequality, we have $c_{v_i v_{i+1}} \leq c'(Z_{v_I v_{i+1}})$
   so we have $c'(\overline{Z}) \leq c'(z) = 2 * OPT$

5.3 discussion

1. A α-approximation algorithm with $\alpha \geq 1$
   For a minimization probelm, is a efficient algorithm is efficient algorithm that on every cases
   it will ruturn a solution of cost $\leq \alpha *$ (optimal)
   So algorithm for steiren tree is a 2-approximation algorithm

5.4 Computational complexity - the class P

1. P
   is the set of all problems that can be solved by polytime algorithm

2. This includes

   - SP, MST, MCA
   - Is composite
   - Factoring: by using is composite
   - Is prime?

3. Algorithm for factoring: Eratoochhenes Sieve
   consider all $x \in [2, \sqrt{n}]$, see it devide n
   if so then n is composite, and x, n/x is a factorzation of n
   not poly time

4. Decision problem
   A problem with yes/no answer

5.5 Polynomial-Time Reduction

1. Definition
   Let probelm A, B we say A reduct in polytime to B, denote $A \leq_p B$,
   if we can solve A using algorithm of B polynumber of time

2. note
   if $B \in P$, $A \in P$
   if $A \notin P$, $B \notin P$

6 Week 6

6.1 Computional complexity model

1. Verifier V for Dec problem P
   is an algorithm that take two input

   - x: a feasible solution of P
   - y : a certificate

   Output: Yes/No which saticsfied

   - x is a Yes instance of P, then $\exists y$ such that V(x, y) = YES
   - x is a No instance of P, then $\forall y$, V(x ,y) = No

2. NP
   If $p \in NP$, then

   - $\exists$ polynomial p
   - a verifier V

   Such that
   $\forall$ yes instance of x of P, $\exists$ a certificate y with $|y| \leq p(|x|)$
   means the number of bits to specify y is poly less than bit need to specify x
   s.t. V(x, y) = yes
   Generaly speaking, a NP problem's yes instance will have a shorter certificate

3. Claim 6.1 $P \subseteq NP$
   if a dec problem $P \in P$, $P \in NP$

4. More defs
   A probelm B is

   - NP - hard: if $X \leq_p B, \forall X \in NP$
   - NP- complete: if $B \in NP$, and B is NP hard

5. Proves

   - find a NP-hard problem B
   - show $B \leq_p Y$, if so, Y is np-hard
   - if y is NP, then Y is NP complete

7 week 7

7.1 NP completeness of Set cover and Steiner tree

1. Cook-kevin theorem
   3-SAT is NP-complete

2. Set cover $\overline{\cap}$
   Given universe U and a collection δ of subset of U, integer $k \geq 0$
   are there k set in δ which union is U?

3. Theorem: 3-SAT $\leq_p \overline{\cap}$

   - vertex cover
     let G = (V, E) be a undirected graph with
     U = E, $\delta = \{\delta(v), v \in V\}$
   - theorem 7.2: 3-SAT $\leq_p$ vector cover $\leq_p \overline{\cap}$
   - Proof
     - Let $F = \cap C_i$, where $C_i = y_{i1} + y_{i2} + y_{i3}$ (3 y only)
     - create a graph based on the rule that
       every y_i is a vertex (y_{ij}, i)
       edges are created between $(y_{ij}, i), (y_{ab}, a)$ if
       i = a or $\overline{y_{ij}} = y_{ab}$
     - show F is satisifiable $\iff$ the graph is a vertex cover with size $\leq 2m$ (k = 2m)
       * =>
         every C_i must have one y_i be true
         let $\delta = \{\delta(y_{ij}) : y_{ij}$ false $\}$ and this have maximum 2m vertices
         and this will cover all edges in vector cover since
         all edges in triangle will be cover and y_{ij} and $\overline{y_{ij}}$ can't be both false or true
       * <=

4. Theorem 7.3 Set cover $\leq_p$ Steiner tree
   a set cover of size $\leq k \iff$ a steiner tree of cost $\leq n(k+1) + k, n = |U|$

8 Week 8

8.1 a-approcximation algorithms

1. Designing steps

   - Come up with a LB on OPT
   - Design an polytime algorithm and show that it return a cost $\leq a * LB \leq a * OPT$

2. Theorem 8.1
   LPs can be solved in polytime

3. LP rounding algorithms
   this is a a approximation problem

4. Promed - Dual algorithms
   we know every feasible solution y to D has a value $\leq OPT_{(p)}$
   so we can think of designing an algorithm that contract an interger solution x and dual solution y
   show $cost(x) \leq a * (value(y))$

5. Some notation for LP relaxation
   $\lambda = \{S \subseteq V : S \cap T \neq \emptyset, T - S \neq \emptyset\}$
   $\delta(s)$ is cut of s

6. Duality

   - Weak Duality
     Let x be a feasible solution to P, y be a fesible solution to D
     then $\sum_{e \in E} c_e x_e \geq \sum_{s \in \lambda} y_s$
   - Strong duality
     if x and y are optimal, then $\sum_{e \in E} c_e x_e = \sum_{s \in \lambda} y_s$
   - Complementary slackness condition
     - $x*_e > 0$, $\sum_{s \in \lambda, e \in \delta(s)} y*_s = c_e$
     - $y*_s > 0$, $\sum_{e \in \delta(s)} x*_e = 1$

8.2 Network connectity problem

1. Modeling Framework

   - define a new $f(s) : 2^v \to \{0, 1\}$
     where $f(s) = 1$ for $s \subseteq v$ shows that a feasible solution must include a edge from $\delta(s)$ (cut of s)
   - f-connectivity problem
     find a min-cost set of edge F such that
     $F \cap \delta(s) \neq \emptyset, \forall s \subseteq V, f(s) = 1$

2. LP-relaxation for f-connectivity problem
   let $\lambda = \{S \subseteq V : f(s) = 1\}$
   LP: min $\sum_{e \in E} c_e x_e$

   - $\sum_{e \in \delta(s)} x_e \geq 1, \forall S \in \lambda$
   - $x_e \geq 0, \forall e \in E$

   Dual: max $\sum_{S \in \lambda} y_S$

   - $\sum_{s \in \lambda : e \in \delta(s)} y_s \leq c_e, \forall e \in E$
   - $y_s \geq 0, \forall S \in \lambda$

3. $\{0, 1\}$ proper function: $f : 2^v \to \{0, 1\}$

   - f(v) = 0
   - f(s) = f(v-s), $\forall S \subseteq V$
   - $\forall A, B \subseteq V, A, B \neq \emptyset, A \cap B = \emptyset$
     $f(A \cup B) = 1 \to f(A) = 1$ or $f(B) = 1$

9 Week 9

9.1 Primal-dual algorithm

1. Remainder
   $\lambda = \{S \subseteq V : f(s) = 1\}$

2. Violated set
   At any stage, given a infeasible solution F, S is a violated set,
   then $f(s) = 1, F \cup \delta(S) = \emptyset$

3. Minimal Violated set (MVSs)
   let $V = \{S \in \lambda : f(S) = 1, F \cup \delta(S) = \emptyset, \forall T \subset S, T \neq S,$T is not violated$\}$

4. Lemma 8,2
   Given a set F od edges, the MVS are
   $\{S \subseteq V : Sisacomponentof(V, F)andf(s) = 1\}$

5. Corollary 9.1
   $F \subseteq E$ is feasible $\iff$ every connected component C of (V, F) has $f(C) = 0$

6. Primal Dual Algorithm

   - Initializa $F = \emptyset$, $y_s = 0, \forall S \in \lambda$, $t = 0$
     with $V* = \{S \subseteq V$:S is a component of (V, F) s.t.$f(S) = 1\}$ which is the MVSs
   - While $V* \neq \emptyset$
     raise y_s uniformly at the same rate $\forall S \in V*$, until some edge $e \in \delta(S)$ goes tight for some $S \in V$
     update $F = F \cup \{e\}$, update $V*$ due to the change of F
   - Reverse delte
     let $F = \{e_k \dots e_1\}$ where e is the last one instered
     if $F - \{e_i\}$ is feasible, set it to F
   - Return

7. $\delta_Z(S) = \delta(S) \cap Z$

8. Lemma 9.2
   At anypoint in algorithm $\sum_{S \in V*} |\delta_f(s)| \leq 2|V*|$

9. Theorem 9.3
   Let F be the return, y be a feasible dual solution such that $C(F) \leq 2 \sum_{S \in \lambda} y_S$
   hence it is a 2-approximation algorithm

10 Week 10

10.1 Mar 15: Set cover - primal-dual and greedy algorithms

1. Review set cover
   Given universe U, and $\lambda = \{S_i\}, S_i \subseteq U$, with W_{S_i} represent the weight of each set
   find the minimum-weight collection of set union = U

2. Special case
   if $W_S = 1$ it is NP-hard, even vector cover

3. LP-relaxation for set cover and dual
   S is element of λ, e is element of U

   - P:
     min $\sum_S W_S x_S$
     s.t. $\forall e \in U, \sum_{S:e\in S} x_S \geq 1$
     $x \geq 0$
   - D:
     max $\sum_e y_e$
     s.t. $\forall S \in \lambda, \sum_{e\in S} y_e \leq W_S$
     $y \geq 0$

4. Complentary slackness

   - $x_S > 0 \to \sum_{e\in S} y_e = w_S$
   - $y_e > 0 \to \sum_{S:e\in S} x_S = 1$

5. B-approximation Algorithm
   $B = max|\{S \in \lambda : e \in S\}|$

   - Inilize $y_e = 0, \forall e \in V, \delta = \emptyset$
     $N = U - \cup_{S\in\delta} S$
   - While $N \neq \emptyset$
     Pick some $e \in N$, raise y_e until $\sum_{e\in S} y_E = W_S$ for some S such $e \in S$, make $\delta = \delta \cup S$
     update N
   - Return δ

6. Theorem 10.1: The above algorithm is B-approximation algorithm

7. Greedy algorithm for set cover
    - Intilize $y_e = 0, \forall e \in V, \delta = \emptyset$
      $N = U - \cup_{S\in\delta} S$, (just like b-appro), $t = 0$
    - while $N \neq \emptyset$
      raise y_e uniformly at some rate unitl $\sum_{e\in S\cap N} y_e = w_S$
      $\delta = \delta \cup \{S\}, N = N - S$
    - return δ
8. Analysis for greedy
    - y constructed is not necessaryly a feasible solution
    - let $\Delta = max_{S\in\delta}|S|$
    - for a integer k $H_k = \sum_{i=1}^{k} \frac{1}{i}$
9. Theorem 10.3
    - $y' = \frac{y}{H_\Delta}$ is a feasible dual solution
    - greedy is a H_Δ approcimation algorithm
10. Theorem 10.2
    For every set S, $\sum_{e\in S} y_e \leq H_\Delta * w_S$

10.2 Mar 17: Set cover - LP rounding algorithms, uncapacitated facility location

1. LP rounding algorithm Observations Let $x*$ be optimal solution for SC-P

   - if $x_S* \geq \frac{1}{c}$ and we set $x_S = 1$ in an integer solution
     then cost increase by a factor of at most c
   - $\forall e \in U, \exists S, x_S* \geq \frac{1}{B}$
   - Combine the first 2 we get
     set x_S to
     - 1, if $x * S \geq \frac{1}{B}$
     - 0, if not

2. O(ln n)-app algorithm, n = —U— Idea: interpret $x*_S \in [0, 1]$ as probability of weather S pick it or not
   we may have 2 error

   - we may not cover all element
   - can only say expected cost is bounded

   expected $= \sum_{S\in\lambda} P = \sum_{S\in\lambda} w_S x_S* = OPT_{LP}$

3. Deal with first error

4. Uncomactated facity location(UFL)
   Given:

   - a complete bipartite graph, $G = (V = F \cup C, E)$
   - every $i \in F$ have a opening cost $f_i \geq 0$
   - each edge $e = ij \in E$ have a connection cost $c_{ij} \geq 0$, which is the cost of assign client j to faclity i

   Goal: choose a set of $F' \subseteq F$ of facilities to open, and a assignment $\{ij\} = E' \subseteq E$, where $i \in F'$
   to minimize $\sum_{i \in F'} f_i + \sum_{i \in F', j \in c} c_{ij}$

5. LP relaxization
   define the following

   - y_i indicate if facility f is open
   - x_{ij} indicate if assignment between i and j is there

   min $\sum_{i \in F} f_i y_i + \sum_{j \in C} \sum_{i \in F} c_{ij} x_{ij}$ (UFL-P)
   s.t.

   - $\sum_i x_{ij} \geq 1, \forall j \in C$
   - $x_{ij} \leq y_i$
   - $x, y \geq 0$

   max $\sum_j a_j$ (UFL - D)
   s.t.

   - $a_j - B_{ij} \leq c_{ij}$
   - $\sum B_{ij} \leq f_i$
   - $a, B \geq 0$

11 Week 11

11.1 Mar 22: UFL - general assignment costs

1. Economic

   - α_j= amount c in willing to pay to get connected
   - First of all, assume no open cost, then each client c will pay lowest assignment cost $min_{i\in F}(c_{ij})$
   - After there is open cost of a protion $B_{ij} \geq 0$ of it's opening cost f_i to client j. the opening cost satisified $\sum_j B_{ij} \leq f_i$ (no overcharge allowed)
   - So the cost for a j has to pay is $c_{ij} + B_{ij}$

2. LP and dual for eco

   - Max $\sum_{j\in C} min_{i\in F}(c_{ij} + B_{ij})$
     s.t.
     $\sum_{j\in C} B_{ij} \leq c_i$
     $B \geq 0$
   - min $\sum_{j\in C} a_j$
     $a_j \leq c_{ij} + B_{ij}$
     $\sum_{j\in C} B_{ij} \leq f_i$
     $B \geq 0, a \geq 0$

3. Complementary Slackness condition

   - $x_{ij} > 0 \to a_i = c_{ij} + B_{ij} \to a_j \geq c_{ij}$
   - $y_i > 0 \to \sum_{j\in C} B_{ij} = f_i$
   - $a_j > 0 \to \sum_{i\in F} x_{ij} = 1$
   - $B_{ij} > 0 \to x_{ij} = y_i$

4. General assignment costs
   let $(x*, y*)$ be solution for $UFL - P$, $(a*, b*)$ be opt solution for UFL-D
   OPT_{LP} be optimal value
   let $F_j* = \{i : x_{ij}* > 0\}$

5. Observations
   $c_{ij} \leq a_j*, \forall i \in F_j*$

   - Suppose we find $F' \subseteq F$ such that $F' \cap F_j* \neq 0, \forall j$
     cost of solution that assigns each client c to some facity in $F' \cap F_j*$ is at most
     $\sum_{i \in F'} f_i +$ total client assignment cost $\leq \sum_{i \in F'} f_i + \sum_{j \in C} a_j = \sum_{i \in F'} f_i + OPT_{LP}$
   - Find mi-cost set F' satisified set cover
     for each facility i, set weight is f_i
     that cover all client j where $i \in F_j*$
     $U = C$
   - $y*$ is feasioble to LP relaxiation for set cover in last observation
   - using greddy algorithm for set cover, we get a $H_n + 1$ approximation for UFL, where $n = |C|$

11.2 Metric UFL

1. Definition
   all c_{ij} satisified triangle inequality
   which is for $i, i' \in F, j, j' \in C, c_{i'j} \leq c_{ij} + c_{ij'} + c_{i'j'}$

2. Clustering Algorithm to find c'

   - Let L be list of client in c sorted in up order of a_j*
   - Initialize $c' = \emptyset$
   - While $L \neq \emptyset$
     - Let j: first client in L; set $c' = c' \cup \{j\}, L = L - \{j\}$
     - Forall $k \in L$ such that $F_k* \cap F_j* \neq \emptyset$, set $L = L - \{k\}$, neighbor(k) = j
   - Return c'

3. Algorithm for metric UFL

   - get c'
   - set $F' = \emptyset$
   - for each $j \in C'$
     - Open $i \in F_j*$ with smallest f_i, add this to F'
     - Assign j to i, and assign every $k \in C - C'$ with neighbor(k) = j to i
   - Return F', clienet assignment in step 3

4. lemma 11.2
   if $j \in C'$, then it is assigned to facility i such that $c_{ij} \leq a_j*$
   if $k \in C - C'$, then it is assigned to a facility i such that $c_{ik} \leq 3 * a_k*$

5. lemma 11.3
   cost for open facilities is at most $\sum_i f_i y_i*$

6. Theorem 11.4
   we have a 4 appriximation algorithm

12 Week 12

12.1 Mar 29: Solution methods - relaxations

1. Relaxation
   Assume We have P : max f(x) s.t. $x \in X$
   With R : max g(x) s.t. $x \in G$
   then R is a relaxation of P if
   - $X \subseteq G$
   - $f(x) \leq g(x)$
2. Notation
   Let x_1 be opt solution to P
   Let x_2 be opt solution to R
3. Theorem 12.1
   $OPT_R \geq OPT_P$
4. Theorem 12.2
   Suppose $f = g$, then if $x_2 \in X$, then it is a optimal solution for P too
5. Lagragian relaxation
   Suppose P : max $c^\intercal x$ s.t. $Dx \leq d, x \in X$
   Let $\lambda \in R_+^m$ consider
   $LR(\lambda)$: max $c^\intercal x + \lambda^\intercal(d - Dx)$ s.t. $x \in X$
   $=$ max $\lambda^\intercal d + (c^\intercal - \lambda^\intercal D)x$ s.t. $x \in X$
6. Lemma 12.3
   $LR(\lambda)$ is a relaxation of P $\forall \lambda \geq 0$
7. Example with MST
   - Given
     a undirected $G = (V, E)$, weight $\{w_e\}$, degree bounds $\{b_v \geq 0\}$
   - Goal:
     find a ST of maximum weight s.t. every node v has degree $\leq b_v$
8. Incidence vector
   Let $F \subseteq E$, X^F of F is the $\{0, 1\}$ vector in R^E given by
   $X_e^F = 1$ if $e \in F$, 0 otherwise

9. Convex
   A set $Z \subseteq R^n$ is convex if $\forall x, y \in Z, \forall \lambda \in [0, 1]$
   $\lambda x + (1 - \lambda)y$ is also in z
   Note: $\emptyset$ is a convex set
   If $\{Z_i\}$ are all convex set, the intersection of them is convex too

10. Convex hull
    denoted $conv(Z)$ is the smallest convex set contain Z
    if c is convex and $conv(z) \nsubseteq C$ or $C \subset conv(z)$ then $Z - C \neq \emptyset$

11. Theorem 12.4
    Let P be: max $c^\intercal xs.t.Ax \leq b, xinteger$
    Let A, b be rational, then

    - The conv(z) is a polyhedron (a set of $\{x : Mx \leq d\}$)
    - Consider R: max $c^\intercal xs.t.x \in conv(z)$, then
      - (R) is a ralaxation of (P)
      - (R) is an LP
      - $OPT_R = OPT_P$

12. Lemma 12.5
    Suppose $Z = \{\epsilon_i\} \subseteq R^n$
    then conv(Z) is a polyhedron

12.2 Branch and Bound, knapsack problem

1. Lemma 12.6
   Let $Z \subseteq R^n$
   P: max $c^\intercal x$ s.t. $x \in Z$
   R: max $c^\intercal x$ s.t. $x \in conv(Z)$
   $x*$ be opt solution for P, then it is also a opt solution for R

2. theorem 12.7
   Suppose $conv(X_1)$ is a polyhedron,
   then $\epsilon_{LD} = max c^\intercal x$ s.t. $Dx \leq d, x \in conv(X_1)$
   In particular if

   - X_1 is finite, or
   - $X_1 = \{x : Ax \leq b, xint\}$

   where A, b are rational
   then $conv(X_1)$ is a polyhedron, and the above explanation for ϵ_{LD} holds

3. Branch and Bound Metohd
   goal: solve the optimalzation problem P: max $f(x)$ st.t. $x \in X$
   the arguments are like $Bnb(Q, LB)$ where Q is the problem, LB is the lower bound

   - Solve a relaxation (R) of Q, (Assume R is not unbounded)
     If R is infeasible, return Q is infeasible
     Otherwise, Let $\epsilon_R = OPT_R$, $x^{(R)}$ be optimal solution for R
   - If $\epsilon_R \leq$ LB, then stop, LB is the best
   - Otherwise, use $x^{(R)}$ to find opt solution to Q
     If we can use 12.2, then use it
     If not, Branch: Partition X_Q into union of X_i
     and recursively call BnB to find best solution for each X_i
     update LB when we find a feasible solution to P and always maintain the best solution found so far
   - Return LB at the end
   - Theorem 12.8, a algo to solve LP for knapsock
     List item $\{1 \ldots n\}$ in decreasing order of $P_i = v_i/a_i$
     set each x_i as large as possible, and find the x_l such we switch from 1 to 0
     and branch 2 of $x_{l+1} = 1$ and $x_{l+1} = 0$