Lecture 11: Minimum Spanning Trees & Cone Programming

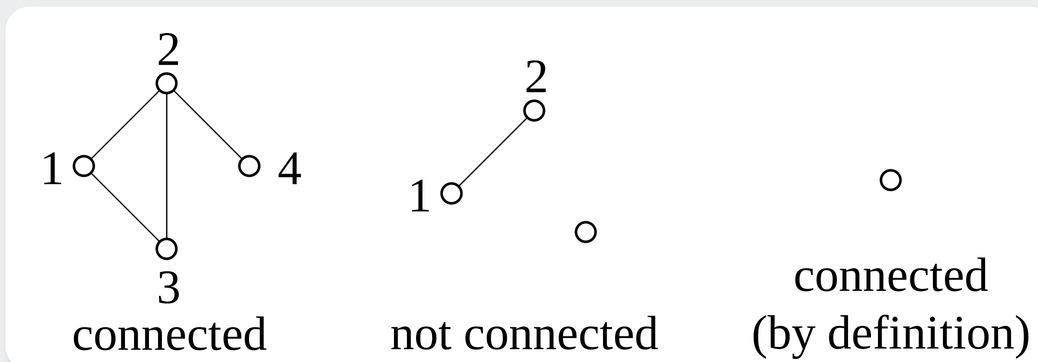
Prof. Krishna R. Pattipati
Dept. of Electrical and Computer Engineering
University of Connecticut
Contact: krishna@engr.uconn.edu; (860) 486-2890

Outline

- Review of relevant theory
- Why solve the minimum spanning tree problem?
- Three basic algorithms
  - Kruskal (1956)
  - Jarnik-Prim-Dijkstra (1930, 1957, 1959)
  - Bor'uvka (1926) … a distributed algorithm
- Application to centralized communication network design problem
- Introduction to Cone Programming

Review of Relevant Graph Theory

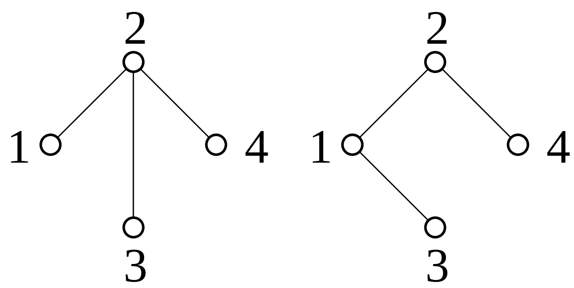
- Undirected graph $G = (V, E)$
  - V = set of vertices (nodes)
  - E = set of edges (arcs)
- A graph G is connected if, for every node i, $\exists$ a path $(i = v_1, v_2, \ldots, v_l = j)$ to every node j



- Not connected $\Rightarrow$ can find two sets of nodes with no edges between them
- Basic result:
  - For a connected graph G, if $X \in V$ is a nonempty subset of V, then $\exists$ at least one edge $(i, j) \ni i \in X$ and $j \in \bar{X} = (V - X)$
  - You can think of the partition of vector set V into X and $\bar{X}$ as a cut in graph G and the edge (i, j) crosses the cut since it is incident on X (one end in X the other in $\bar{X}$)

Spanning Tree and Forest

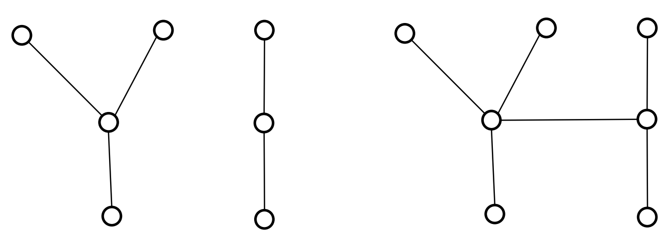
- A tree is a connected graph with no cycles (loops, circuits) ⇒ $n - 1$ arcs (edges)
- **A spanning tree of a connected graph *G* is a tree and contains all the nodes of *G***



Spanning tree of graph *G*

- # of nodes = n
- # of edges = $n - 1$
- There exists a single path between every pair
- Adding an edge results in exactly one cycle
- Deleting an edge makes the tree disconnected

- **A forest (fragment) is a node-disjoint collection of trees**



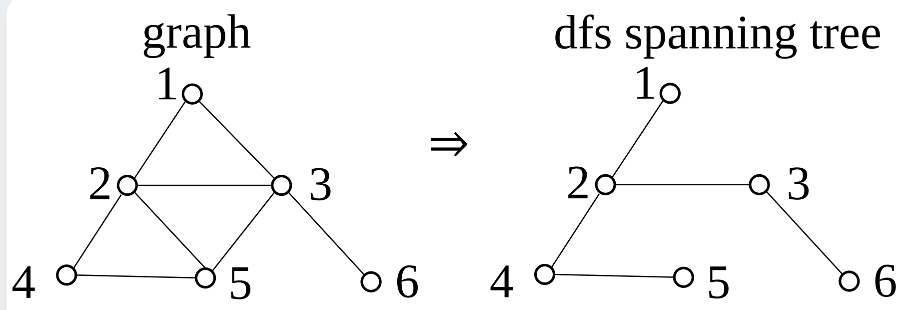
forest: set of trees　　spanning tree

How to construct a Spanning tree?

- How to construct a spanning tree or how to check for the connectedness of a graph?
  - DFS: select an edge $(i, j) \ni i$ was visited most recently … stack or LIFO or recursion. Can also get pre- and post-order traversal
  - BFS: select an edge $(i, j) \ni i$ was visited least recently … queue
- Depth-first generation of spanning tree: call dfs(i)

```
∀ vertex, initialize pre-visit to null
procedure dfs(i)
        pre-visit(i)
        for (i, j) ∈ out(i) do
                if not visited(j)
                        parent(j) =i
                        dfs(j)
                end if
        end do
        post-visit(i)
```

$O(m)$ complexity

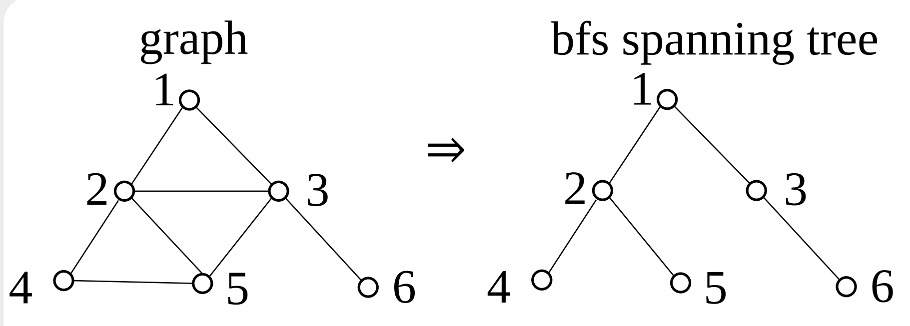


Breadth-first generation of a Spanning tree

- Breadth-first search generation of spanning tree: call bfs(1)

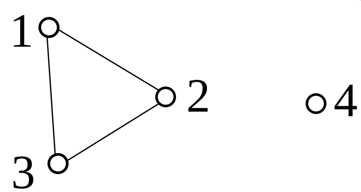
$\forall$ vertex, initialize bfs-visit to null
procedure bfs(1)
    $queue = \{1\}$
    while $queue$ not empty do
        $i = queue[1]$; $queue = \{2, \ldots\}$
        bfs-visit(i)
        for $(i, j) \in \text{out}(i)$ do
            if not visited j & $j \notin queue$
                $queue = queue \cup \{j\}$
            end if
        end do
    end do

$O(m)$ complexity



⇒ For every **connected** graph G with n nodes and m arcs $\exists$ a spanning tree, where $m \geq n - 1$

⇒ G is a tree *iff* number of edges of the tree, $m = n - 1$ **and connected**



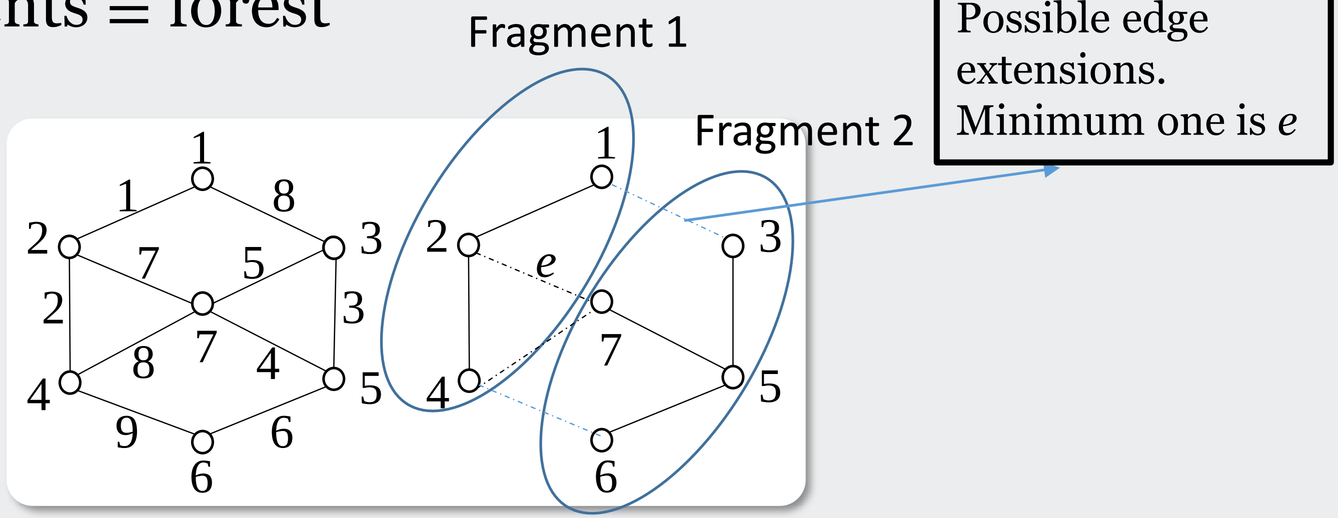
⇒ need connectedness for it to be a tree

Minimal Spanning Tree (MST) Problem

- **Given an undirected connected graph G, each of whose edges has a real-valued cost c_{ij},find a spanning tree of the graph whose total edge cost is minimum**
- Can do for directed or undirected graphs … we will consider undirected graphs only
- Why solve this problem?
  - Arises as a sub-problem in communication network design
    - Connecting terminals to a specified concentrator (switching node) via a multi-drop link
    - Connecting concentrator to a central processing facility
- Want minimum cost connection subject to constraints on:
  - Delay (or flow) on each link
  - Reliability ⇒ alternate paths or not more than a specified number of terminals be disconnected if a link fails
    - Problem is much more involved than MST (in fact, it is NP-hard!)
      - MST forms a starting point for design
      - We will come back to this later
    - Also useful in simplex-based network flow algorithms
      - Recall for network flows, bfs is a spanning tree. See Bersekas' book

Basic Idea of all MST Algorithms

- Incremental construction edge by edge via the greedy method ⇒ do the best thing at every step
- **"Smallest edge first strategy w/o forming cycles"**
- Any sub-tree of a MST will be called a fragment
- Set of fragments ≡ forest



- Main result:
  - Given a fragment F, let $e = (i, j)$ be a minimum weight edge from F where node $j \notin F \Rightarrow F$ extended by edge e and node j is a fragment (i.e., part of MST)

Proof of main result

- Denote by T the MST of which F is fragment
- If $e \in T$, we are done; so, assume otherwise
- Then, there is a cycle formed by e and the edges of T
- Since $j \notin F$, there must be some edge $e' = (i', j)$ that belongs to the cycle, to T and to F
- Deleting (i', j) from T and adding (i, j) to T results in a spanning tree $T' \ni$ cost of $T' \leq$ cost of T

  $\Rightarrow T'$ is an MST

  $\Rightarrow$ So, F extended by e must be part of MST

• Three Classical Algorithms

- Kruskal (1956)
- Jarnik-Prim-Dijkstra (1930, 1957, 1959)
- Bor'uvka (1926) … a distributed algorithm

Three Classical Algorithms

- Kruskal's algorithm
  - Start with each node as a fragment
  - Successively combine two of the fragments by using the edge that has minimum weight and when added does not result in a cycle
- Jarnik-Prim-Dijkstra
  - Select an arbitrary node as a fragment
  - Enlarge the fragment by successively adding a minimum weight edge
- Bor'uvka
  - For every fragment, select a minimum cost edge incident to it
  - Add it to the fragment and inform the fragment that lies at the other end of this edge .... Can do it in a distributed way!

- You can think of these algorithms as edge-coloring processes
  - Blue ⇒ part of MST or accept
  - Red ⇒ not part of MST or reject

Kruskal's algorithm (forest algorithm)

- Sort edge weights in non-decreasing order .... Possibly heaps?
- Using the sorted list, include $e = (i, j)$ if it does not form a cycle (color it blue)
- If it does, discard the edge (or color it red)
- Stop when all $m = (n - 1)$ edges (tree) have been included or all edges have been examined

  $\Rightarrow$ Minimum spanning forest (set of fragmented trees)
- Crude version of Kruskal

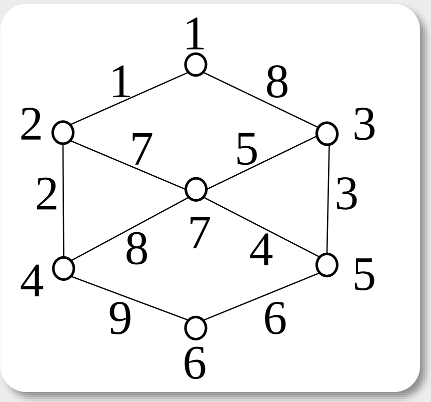
$T = \emptyset$
while $|T| < n - 1$ & $E \neq 0$ do
    e = smallest edge in E
    $E = E - \{e\}$
    if $(T \cup \{e\})$ has no cycle
        $T = T \cup \{e\}$
    end if
end do

- Two hurdles:
  - Sorting m elements requires $O(m \log m)$
    - May be too much work since need only $(n - 1)$ edges
    - Time for heaps??
  - How to test for cycles easily
    - In other words, both ends of the current edge being colored belong to the same fragment

Resolving the two hurdles

- We resolve the first problem by forming a heap
  - $O(m)$ computational steps
  - Finding next minimum takes $O(\log m)$ steps, assuming a binary heap
  - If we do this k times, need $O(k \log m)$ steps

    $\Rightarrow$ total = $O(m + k \log m)$ computation for sorting
- We resolve the second problem by maintaining fragments in the form of subsets of nodes
  - Add a new edge by forming union of two relevant subsets
  - Check for cycle formation by invoking FIND twice to check if two vertices of the edge belong to the same tree (subset, fragment)
- Example

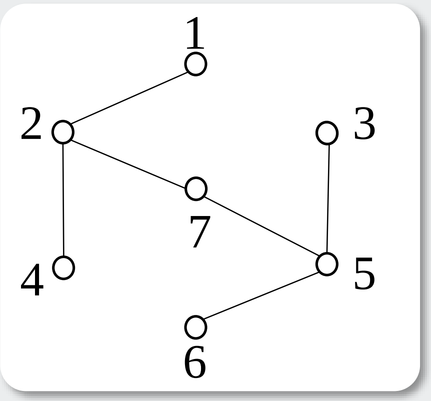


- {1} {2} {3} {4} {5} {6} → {1,2} {3} {4} {5} {6} {7}

→ {1,2,4} {3} {5} {6} {7} → {1,2,4} {3,5} {6} {7}

→ {1,2,4} {3,5,7} {6} → discard edge (3,7)

→ {1,2,4} {3,5,7,6} → {1,2,4,3,5,7,6} *done!!*

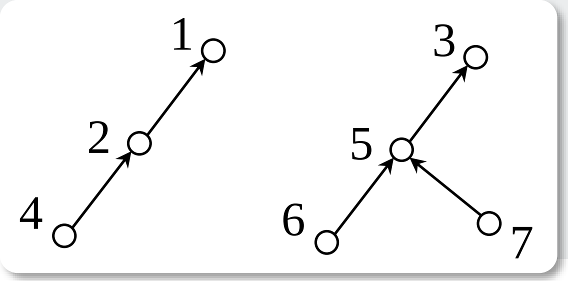


Efficient storage and sorting procedures

- Need efficient methods for sorting fragments (subsets or subtrees)
- Need efficient UNION & FIND procedures

• We can accomplish both of these objectives by storing fragments as rooted trees

- Nodes of the tree are elements of the fragment
- Each node i of the tree has a parent pointer p_i
  - Root node $\begin{cases} \text{no pointer} \\ \text{pointer to } (-\#\text{of elements in the tree})^{**} \\ \text{pointer to (height of the tree or rank)} \end{cases}$
- To carry out FIND(i), we follow parent pointers from i to the root of the tree containing i and return the root
- So to find cycle:
  - If FIND(i) = FIND(j), we have a cycle!!



Efficient storage and union of fragments

- To carry out UNION(x, y), where x and y are roots of subsets

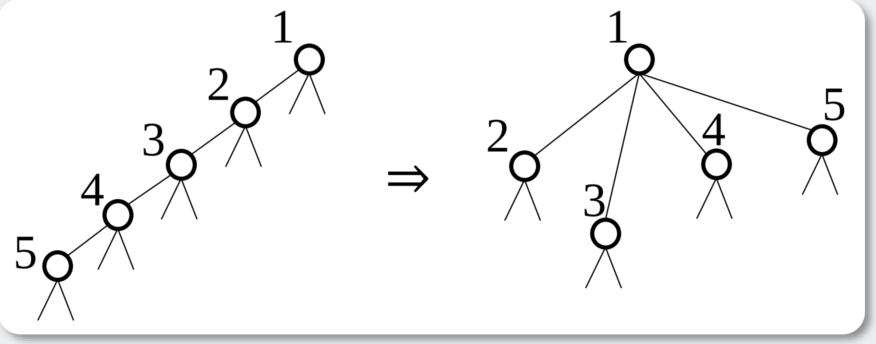
- UNION rank
  - Keep track of rank (height) of trees
  - Do exactly the same as with size except that p_x and p_y denote ranks

if $|p_x| > |p_y|$ then
$p_x = p_x + p_y$
$p_y = x$
else
$p_y = p_x + p_y$
$p_x = y$
end if

- Don't change ranks unless $p_x = p_y$

$\Rightarrow$ make x point to y; $p_x = p_y + 1$

- We can make FIND operation more efficient by a heuristic called path compression
  - When FIND(i) is invoked, after locating root x of the tree, make every node on the path point to the root



- Computational complexity: $O(m\,\alpha(m, n))$ (See Tarjan or Horwitz & Sahni for details) where $\alpha(m, n)$ = inverse of Ackerman's function

Ackerman's function $i, j \geq 1$

$$A(1,j) = 2^j, \quad \forall j \geq 1$$

$$A(i,1) = A(i-1,2), \quad \forall i \geq 2$$

$$A(i,j) = A\big(i-1, A(i,j-1)\big), \quad \forall i,j \geq 2$$

$$\alpha(m,n) = \min\left\{i \geq 1 : A\left(i, \left\lfloor \frac{m}{n} \right\rfloor\right) > \log n\right\}$$

- Note that $A(2,1) = A(1,2) = 4$
- $A(3,1) = A(2,2) = A(1, A(2,1)) = A(1,4) = 2^4 = 16$
- $\alpha(m,n) = \min\{\cdot\} \leq 3, \forall n < 2^{16} = 65{,}536$
- $A(4,1) = A(2,16) = 2^{\text{“big number”}}$ which is very large
- For all practical purposes, $\alpha(m,n) \leq 4$

$\Rightarrow$ Computational complexity $O(3m)$ or $O(4m)$

Overall Kruskal

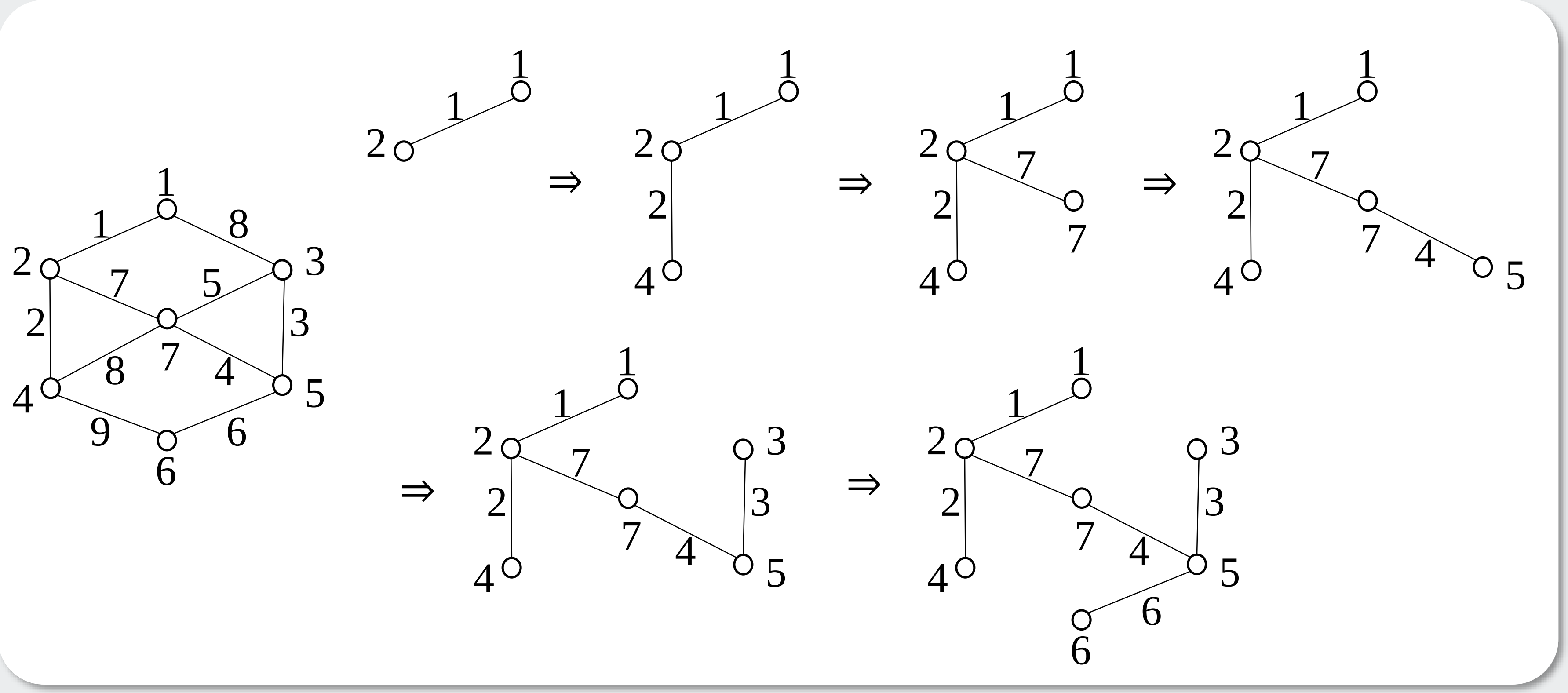
```
set father (parent) array to −1 or rank = 0
form initial heap of m edges
edge_count = tree_count = 0; T ← ∅
while (tree_count < n − 1 & edge_count < m) do
    e = edge(i, j) from top of heap
    edge_count = edge_count + 1
    remove e from heap & restore heap … delete min operation
    r1 = FIND(i); r2 = FIND(j)
    if (r1 ≠ r2) then
        T = T ∪ {e}
        tree_count = tree_count + 1
        UNION(r1, r2)
    end if
end do
```

- Function FIND(i) {does path compression also}

```
if p_i > 0
    p_i = FIND(p_i)
end if
return p_i
```

Jarnik-Prim-Dijkstra Single Tree Algorithm

- Start with a single node as a fragment and repeat the following step $(n-1)$ times
  - **"If T is the current MST generated so far, select a minimum cost edge incident to T and include it in T (or color it blue)"**
- Example



Jarnik-Prim-Dijkstra's procedure

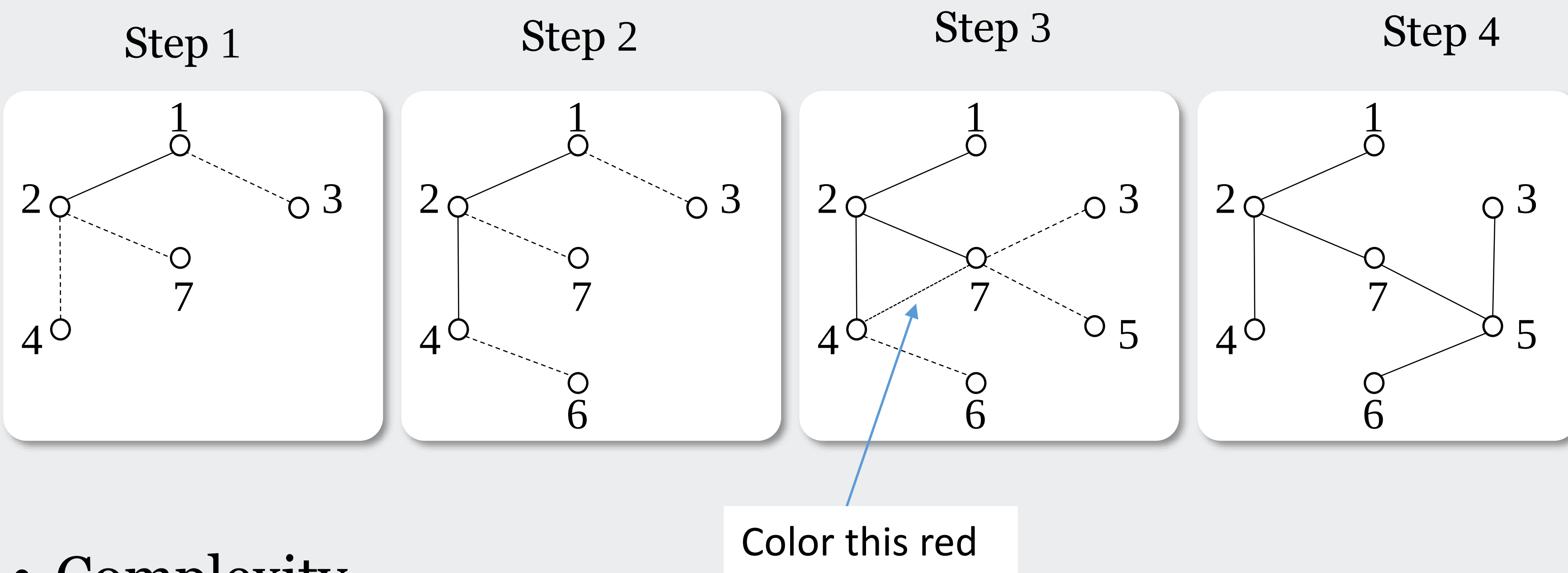
- Suppose T is the MST generated so far
- Find neighbor nodes i to T ∋ an edge is incident to both i and T
- With each neighbor i, associate a light blue edge (k, i)

  ⇒ That is, a minimum-cost edge incident to i and T

  ⇒ Light blue ⇒ candidates for inclusion into T
- Blue and light blue edges together form a tree spanning T and its neighbor edges
- Coloring step
  - From among these candidates, select one, say (k', i'), of minimum cost and include it in the tree

    $\Rightarrow T \rightarrow T \cup \{i'\}$
  - Consider all edges of the form (i', j):
    - If $j \notin T$ & ∄ a light blue edge of the form (k, j), color (i', j) light blue ⇒ potential candidate
    - Else if $j \notin T$ & ∃ a light blue edge of the form (k, j) & $c_{kj} > c_{i'j} \rightarrow$ mark (k, j) red (or discard (k, j)) and mark (i', j) light-blue (or (i', j) is a potential candidate)

Example



- Complexity

$$\left.\begin{array}{l} n-1 \text{ inserts} \\ n-1 \text{ deletes and restores} \\ \max m-n+1 \text{ siftup operations} \end{array}\right\} \Rightarrow \begin{array}{l} \text{run time: } O(nd\log n + m\log n) \\ d = \left\lceil 2+\dfrac{m}{n}\right\rceil \Rightarrow O\left(m \log_{\left\lceil 2+\frac{m}{n}\right\rceil} n\right) \end{array}$$

Heap Implementation

for each node i

adj_list = set of edges incident to i

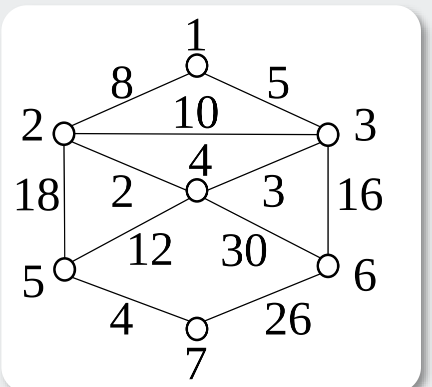
$$\text{blue}(i) = \begin{cases} \text{undefined} & \text{if } i \notin T \cup \{\text{neighbor } T\} \\ \text{light blue edge incident to } i & \text{if } i \in \{\text{neighbor } T\} \\ \text{blue edge} & \text{if } i \in T \end{cases}$$

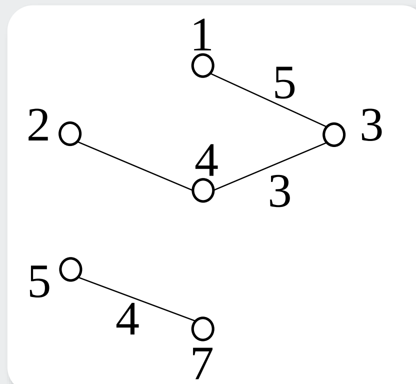
$$\text{cost}(i) = \begin{cases} \infty & \text{if } i \notin T \cup \{\text{neighbor } T\} \\ \text{cost of light blue edge} & \text{if } i \in \{\text{neighbor } T\} \\ -\infty & \text{if } i \in T \end{cases}$$

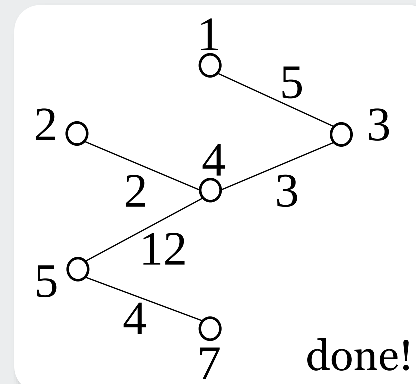
for $i = 1, \ldots, n$ do
  cost(i) = ∞
$h = \emptyset$; $i = 1$
while $i \neq null$ do
  cost(i) = $-\infty$
  for $(i, j) \in adj\_list(i)$ do
    if($c_{ij} <$ cost(j))
      cost(j) = c_{ij}; blue(j) = (i, j)
      if $j \notin h$
        insert j into h
      else
        siftup j
      i= min of heap for which original min was added

Bor’uvka’s distributed algorithm

- Bor’uvka’s distributed algorithm
  - First assume that all edge weights c_{ij} are distinct
  - Start with a set of fragments
  - Each fragment determines its own minimum edge and informs the fragment that lies at the other end
  - The algorithm correctly terminates!!

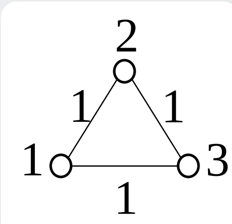






- How does each fragment decide on it minimum weight arc?
  - See P. Humblet, “A distributed algorithm for minimum weight directed spanning trees,” <u>IEEE Trans. On Comm.</u>, vol. COM-31, pp 756-762
- What can go wrong when have non-distinct costs?

  ⇒ Cycles



Proof and algorithm extension

- If all edge weights are different, have a unique MST
  - Suppose non-unique ⇒ at least two MSTs, say T and T'
  - Let $(i,j) = \arg\min \{c_{lm}\}$ and assume $(i,j) \in T$
  - Suppose add (i,j) to T'

    ⇒ Cycle

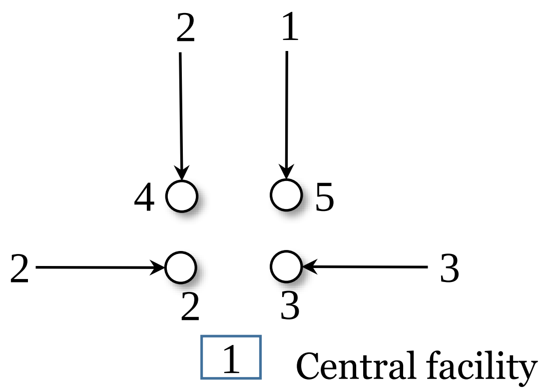
    ⇒ Can throw away an arc (k,l) and get a new spanning tree with less cost

    ⇒ T' not optimal

    ⇒ contradiction
- To extend Bor'uvka's algorithm to non-distinct weight case, do the following:
  - In the case of equal weight, break the tie in favor of an edge with a minimum identity end node and if these nodes are the same, break the tie in favor of an edge whose other node has a smaller identity
  - In this case, we are guaranteed a unique MST

Application: communication network design

- We will illustrate the MST application via a simple example

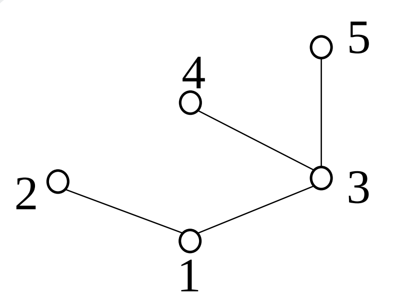


c_{ij} ($i \downarrow$, $j \rightarrow$)

i \ j	1	2	3	4	5
1	–	3	3	5	10
2	3	–	6	4	8
3	3	6	–	3	5
4	5	4	3	–	7
5	10	8	5	7	–

$f_{ij} \leq 5, \forall i, j$

- Problem w/o constraints is MST



Cost = 14

Prim's version

- Step 0: initialize each node i with a weighting factor $w_i \ni$
  - $w_1 = 0; \;\; w_i = -\infty, \;\; \forall i \neq 1$
  - $t_{ij} \leftarrow c_{ij} - w_i \Rightarrow t_{ij} = \infty \ni i \neq 1$
  - t_{ij} = saving gained by removing the central connection and creating a link connection
  - {initially then all $t_{ij} = \infty$ except t_{1j} representing the cost of connecting each node to the center}
  - find $\min\{t_{ij} = t_{qm}\}$
- Step 1: {in the example, connect 2 or 3 … Say, we select (1,2)}
- Step 2: if constraints are not violated

    add link (q, m)

    set $w_m = 0$

    readjust constraints and recalculate all t_{ij}

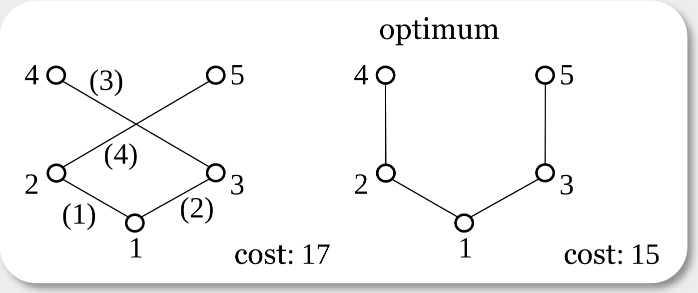
    go back to Step 1

  Else:

    set $t_{qm} = \infty$

    go back to Step 1

  - {add link (3,1), then (4,3), and finally (5,2)}



Kruskal's version and Esau-Williams algorithm

- Kruskal's version:
  - Select minimum cost links one at a time, check for constraints and repeat procedure
  - Ordering: (1,2) (1,3) (4,3) (5,2) … same as Prim … cost = 17
- Esau-Williams algorithm:
  - Step 0: let $t_{ij} = c_{ij} - c_{i1}, \forall i, j$

    {t_{ij} = a measure of difference in cost of connecting node i to node j vs. connecting node i to node 1}

    $$t_{24} = c_{24} - c_{21} = 4 - 3 = 1$$

    $$t_{42} = c_{42} - c_{41} = 4 - 5 = -1$$

    {⇒ node 2 is closer to the center than to node 4 and node 4 is closer to 2 than to the center}

    $$t_{53} = c_{53} - c_{51} = 5 - 10 = -5$$

    $$t_{35} = c_{35} - c_{31} = 5 - 3 = 2$$

    In addition, $t_{21} = t_{31} = t_{41} = t_{51} = 0$
  - Step 1: select min{$t_{ij} = t_{lm}$} and consider connecting i to j

Esau-Williams algorithm - continued

- Step 2: if constraints are not violated

  Add link (l,m)

  Label node l with node m label showing l connected to m

  Reevaluate constraints and update trade-off functions

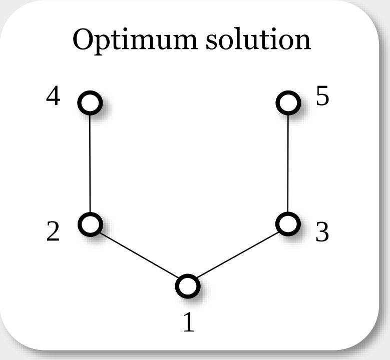
  Go to Step 1

  else

  set $t_{lm} = \infty$

  Go back to Step 1

  end if
- We get optimal solution here
- For details, see:
  - Chandy, K. H and R. A. Russel, "The design of multi-point linkages in a teleprocessing tree network," IEEE T-Comp., vol. C-21, Oct. 1972, pp. 1062-1066
  - A. Kreshnebaum and W. Chose, "A unified algorithm for designing multi-drop teleprocessing network," IEEE T-Comm., vol. COM-22, Nov. 1974, pp. 1762-1772



Variations

- On-line algorithms
  - Maintain a set of blue trees
  - To process an edge, color it blue
  - If this forms a cycle of blue edges, discard a maximum-cost blue-edge on the cycle
  - Complexity $O(m \log n)$
  - See: F. Maffioli, "Complexity of Optimum Undirected Tree Problems: A Survey of Recent Results," Analysis and Design of Algorithms in Combinatorial Optimization, Springer-Verlag, NY, 1981
- Alternative cost structures
  - Can change c_{ij} to any monotonic function of c_{ij}
- How much can you increase/decrease the cost of an edge without affecting the minimality of the spanning tree?
  - Complexity $\leq O(4m)$ … see Tarjan
- Degree constraints at nodes $\Rightarrow$ NP-complete
  - Degree ≤ 2 at each node $\Rightarrow$Hamiltonian path problem

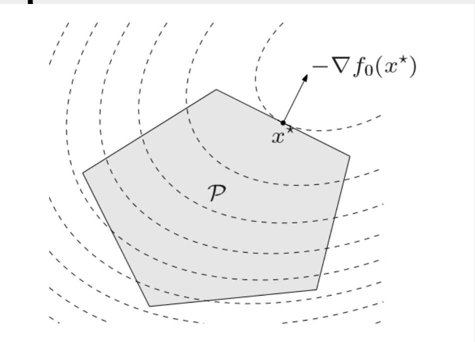
A bit more detailed history

- Late 1940s: Linear programming

$$SLP: \min_{\underline{x}} \underline{c}^T \underline{x} \ s.t.\ A\underline{x} = \underline{b};\ \underline{x} \geq \underline{0}$$

- 1950s: Quadratic programming; minimize a convex quadratic function over a polyhedron

$$QP: \min_{\underline{x}} \frac{1}{2}\underline{x}^T Q\underline{x} + \underline{d}^T \underline{x} + c \ s.t.\ A\underline{x} = \underline{b};\ G\underline{x} \geq \underline{h}$$



- 1960s: Geometric programming

$$GP: \min_{\underline{x}} \sum_{k=1}^{K} c_{0k}\left(\prod_{j=1}^{n} x_j^{a_{0jk}}\right);\ c_{0k} > 0 \text{...posynomial function}$$

$$s.t.\ \sum_{k=1}^{K} c_{ik}\left(\prod_{j=1}^{n} x_j^{a_{ijk}}\right) \leq 1;\ i = 1, 2, .., m;\ c_{ik} > 0$$

$$\xrightarrow{y_j = \ln x_j}$$

$$GP: \min_{\underline{y}} \ln\left(\sum_{k=1}^{K} e^{\left(\underline{a}_{ok}^T \underline{y} + \ln c_{ok}\right)}\right)$$

$$s.t.\ \ln\left(\sum_{k=1}^{K} e^{\left(\underline{a}_{ik}^T \underline{y} + \ln c_{ik}\right)}\right) \leq 0;\ i = 1, 2, .., m$$

- 1990s: Conic programming (second order cone programming (SOCP), semi-definite programming (SDP), robust optimization, etc.)

- Excellent presentation: http://www.robots.ox.ac.uk/~az/lectures/b1/vandenberghe_1_2.pdf

Conic Programming

- Cone: A set C is a cone if $\underline{x} \in C$ implies $\alpha\underline{x} \in C$ for all $\alpha > 0$. A cone that is also convex is a convex cone.
  - Cone, but not convex: $y=|x|$, union of first and third quadrants,...
  - Convex cones

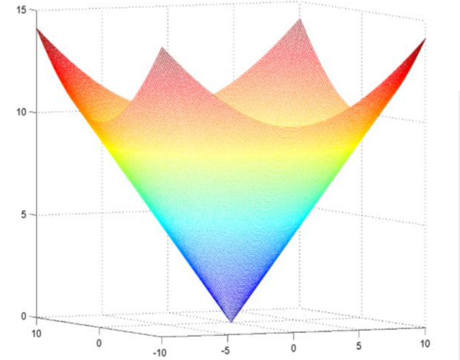
1. $R_+^n = \{\underline{x} : x_i \geq 0, i = 1, 2, .., n\}$

2. $Q_{n+1} = \{(t, \underline{x}) \in R^{n+1} : \| \underline{x} \| \leq t\}$...second order cone

3. $C =$ The set of all poitive semi-definite (SD) matrices, $P \Rightarrow SD\ cone$
   (useful in semi-definite programming (SDP))

4. 2 is special case of 3 with $P = \begin{bmatrix} tI_n & \underline{x} \\ \underline{x}^T & t \end{bmatrix}$

Example of 2:
Ice-cream or
Lorentz cone



- Conic Programming:
  - **Generalized linear programming problems with the addition of nonlinear convex cones**

Varieties of Conic Programs & Applications

- Varieties of Conic Programs
  - Linear programming (LP)
  - Convex Quadratic programming (QP)
  - Quadratically constrained QP (QCQP)
  - Geometric programming (GP)
  - Second order cone programming (SOCP)
  - Semi-definite programming (SDP)

More difficult &
More general

- Applications
  - Signal processing & communications
  - Finance
  - Machine learning
  - Robust control
  - Combinatorial optimization

Second order Cone Programming (SOCP)

- What is SOCP?

$$\min_{\underline{x}} \ \underline{c}^T \underline{x}$$

$$\text{s.t.} \quad A\underline{x} = \underline{b}$$

$$\|C_i \underline{x} + \underline{d}_i\|_2 \leq \underline{e}_i^T \underline{x} + f_i ; i = 1, 2, .., p$$

where $\underline{x}, \underline{c}, \underline{e}_i \in R^n ; C_i \in R^{k_i - 1 \text{x} n} ; \underline{d}_i \in R^{k_i - 1} ; f_i \in R, A \in R^{m\text{x}n}, \underline{b} \in R^m$

- Special cases

$1. k_i = 1 \Rightarrow \underline{e}_i^T \underline{x} + f_i \geq 0 \Rightarrow LP$

2. Convex QP is a special case of SOCP

$$\text{CQP:} \quad \min_{\underline{x}} \ \underline{x}^T Q \underline{x} + \underline{c}^T \underline{x} \ \ s.t. \ A\underline{x} = \underline{b}; C\underline{x} \leq \underline{d}; Q \geq 0$$

$$\text{SOCP:} \quad \min_{\underline{x},t} t + \underline{c}^T \underline{x} \ s.t. \ A\underline{x} = \underline{b}; C\underline{x} \leq \underline{d}; t \geq \underline{x}^T Q \underline{x}$$

(Note: $C_{n+1} = \{(t, Q^{1/2}\underline{x}) \in R^{n+1} : \| Q^{1/2}\underline{x} \| \leq t\}$…second order cone)

- Recall support vector machines is a convex QP ~ SOCP

SOCP and variants

- Special cases and variants

3. Quadratically constrained LP: $\underline{e}_i = \underline{0}$

$$\min_{\underline{x}} \ \underline{c}^T \underline{x}$$

s.t. $A\underline{x} = \underline{b}$

$$\|C_i \underline{x} + \underline{d}_i\|_2 \le f_i \ \Rightarrow \underline{x}^T C_i^T C_i \underline{x} + 2\underline{x}^T C_i^T \underline{d}_i + \underline{d}_i^T \underline{d}_i - f_i \le 0$$

4. SOCP is a special case of SDP... used to approximate integer programs

$$\min_{\underline{x}} \ \underline{c}^T \underline{x}$$

s.t. $A\underline{x} = \underline{b}$

$$\begin{bmatrix} \left(\underline{e}_i^T \underline{x} + f_i\right) I_{k_i - 1} & C_i \underline{x} + \underline{d}_i \\ \left(C_i \underline{x} + \underline{d}_i\right)^T & \underline{e}_i^T \underline{x} + f_i \end{bmatrix} \ge 0, i = 1, 2, .., m$$

Example 1: Robust LP

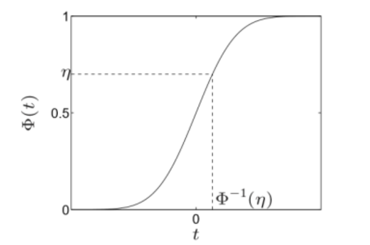
- Inequality constraint with uncertain coefficients

$$\underline{a}_i^T\,\underline{x} \le b_i \text{ where } \underline{a}_i \in \text{ellipsoid E centerd at } \underline{\hat{a}}_i\ , \mathrm{E} = \{\underline{\hat{a}}_i + R_i\underline{u} : \|\underline{u}\|_2 \le 1\}$$

$$\Rightarrow b_i \ge \max_{\underline{x}\in E} \underline{a}_i^T\,\underline{x} = \underline{\hat{a}}_i^T\,\underline{x} + \max_{\|\underline{u}\|_2\le 1} \underline{u}^T R_i^T\,\underline{x} = \underline{\hat{a}}_i^T\,\underline{x} + \|R_i^T\,\underline{x}\|_2$$

$$\Rightarrow \|R_i^T\,\underline{x}\|_2 \le -\underline{\hat{a}}_i^T\,\underline{x} + b_i \Rightarrow \text{ second order cone constraint}$$

- What if $\underline{a}_i$ *is Gaussian*

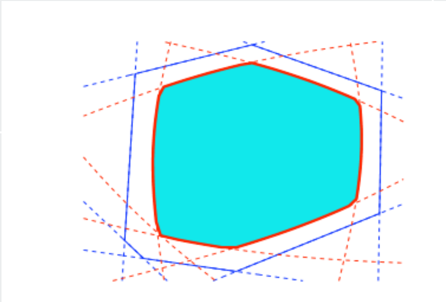


$$\underline{a}_i \sim N(\underline{\hat{a}}_i, \Sigma_i), \Sigma_i = R_i R_i^T \text{ and want } P\{\underline{a}_i^T\,\underline{x} \le b_i\} \ge \eta$$

$$Note : z = \underline{a}_i^T\,\underline{x} - b_i \sim N(\underline{\hat{a}}_i^T\,\underline{x} - b_i, \underline{x}^T R_i R_i^T\,\underline{x}), \sigma_z = \|R_i^T\,\underline{x}\|_2$$

$$P(z \le 0) = \Phi\left(\frac{b_i - \underline{\hat{a}}_i^T\,\underline{x}}{\|R_i^T\,\underline{x}\|_2}\right) \ge \eta; \Phi(y) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{y} e^{-u^2/2}du = Normal\ CDF$$

$$\Rightarrow \Phi^{-1}(\eta)\|R_i^T\,\underline{x}\|_2 \le b_i - \underline{\hat{a}}_i^T\,\underline{x}$$



- Robust LP is a SOCP for $\eta > 0.5$

$$\min_{\underline{x}} \underline{c}^T\,\underline{x}\ s.t.\ \Phi^{-1}(\eta)\|R_i^T\,\underline{x}\|_2 \le b_i - \underline{\hat{a}}_i^T\,\underline{x}, i = 1, 2, .., p$$

Example 2: LP with random cost

- Arises in shortest path problems or network flow problems

$$\underline{c} \sim \mathrm{N}(\hat{c}, \Sigma_c)$$
$$\Rightarrow \underline{c}^T \underline{x} \sim N(\underline{\hat{c}}^T \underline{x}, \underline{x}^T \Sigma_c \underline{x})$$

- Expected cost-variance trade-off (the so-called Markovitz model of risk in portfolio theory when it is formulated as a maximization problem)

$$\min_{\underline{x}} \underline{\hat{c}}^T \underline{x} + \gamma \underline{x}^T \Sigma_c \underline{x}$$
$$s.t.\ \Phi^{-1}(\eta) \| R^T \underline{x} \|_2 \leq b_i - \underline{\hat{a}}_i^T \underline{x}, i = 1, 2, .., p$$

$\gamma > 0$ risk aversion parameter

Example 3: Sparse signal reconstruction

- $\underline{x}$ is a long signal (say, 1000 samples) with *very few* non-zero components (say, 10)

- Want to reconstruct the signal from noisy *m* (say, 100) *noisy* measurements

$$\underline{b} = A\underline{x} + \underline{n};\ \underline{n} \sim N(0, \sigma^2 I_m)$$

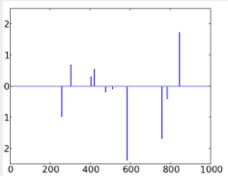
- L_2 *regularization* (Robust least squares)

$$f = \min_{\underline{x}} \| A\underline{x} - \underline{b} \|_2^2 + \gamma \| \underline{x} \|_2^2$$

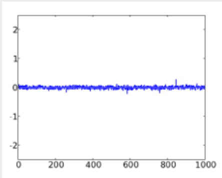
$\gamma >$ 0 regularization parameter

- L_1 *regularization* (LASSO: least absolute shrinkage and selection operator)

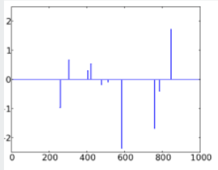
$$f = \min_{\underline{x}} \| A\underline{x} - \underline{b} \|_2^2 + \gamma \| \underline{x} \|_1$$



Original $\underline{x}$



L_2 *reconstruction error*



L_1 *reconstruction*

Recall Primal-dual path following algorithm for LP

- Initialize $\underline{x}_0 > 0,\ \underline{p}_0 > 0,\ \underline{\lambda}_0,\ (\alpha \approx 0.9 - 1)$

for $k = 0,1,2,\ldots k_{\max}$

$t = \underline{p}_k^T \underline{x}_k$

If $t < \varepsilon$, stop

else .....% calculate affine direction

$\mu_k = t / n$

$$\text{solve} \begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ P_k & 0 & D_k \end{bmatrix} \begin{bmatrix} \underline{d}_{xa} \\ \underline{d}_{\lambda a} \\ \underline{d}_{pa} \end{bmatrix} = - \begin{bmatrix} A\underline{x}_k - \underline{b} \\ A^T \underline{\lambda}_k + \underline{p}_k - \underline{c} \\ D_k P_k \underline{e} \end{bmatrix}; D_k = \mathrm{Diag}(\underline{x}_k); P_k = \mathrm{Diag}(\underline{p}_k)$$

Affine direction

calculate

$$\beta_{pa} = \min\left\{1, \alpha \min_{(i:d_{xai}<0)} \left(\frac{-x_{ki}}{d_{xai}}\right)\right\}; \beta_{da} = \min\left\{1, \alpha \min_{(i:d_{pai}<0)} \left(\frac{-p_{ki}}{d_{pai}}\right)\right\}$$

$$\mu_{ak} = (\underline{x}_k + \beta_{pa}\underline{d}_{xa})^T (\underline{p}_k + \beta_{da}\underline{d}_{pa}) / n; \text{Centering parameter } \sigma_k = \left(\mu_{ak} / \mu_k\right)^3$$

When $\mu_{ak} << \mu_k$, it becomes a centering direction

$$\text{solve} \begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ P_k & 0 & D_k \end{bmatrix} \begin{bmatrix} \underline{d}_{x} \\ \underline{d}_{\lambda} \\ \underline{d}_{p} \end{bmatrix} = - \begin{bmatrix} A\underline{x}_k - \underline{b} \\ A^T \underline{\lambda}_k + \underline{p}_k - \underline{c} \\ D_k P_k \underline{e} + \underline{d}_{xa} \circ \underline{d}_{pa} - \sigma_k \mu_k \underline{e} \end{bmatrix}$$

Mehrotra's correction

$$\beta_{p} = \min\left\{1, \alpha \min_{(i:d_{xi}<0)} \left(\frac{-x_{ki}}{d_{xi}}\right)\right\}; \beta_{d} = \min\left\{1, \alpha \min_{(i:d_{pi}<0)} \left(\frac{-p_{ki}}{d_{pi}}\right)\right\}$$

$$\underline{x}_{k+1} = \underline{x}_k + \beta_p \underline{d}_k; \underline{\lambda}_{k+1} = \underline{\lambda}_k + \beta_d \underline{d}_\lambda;\ \underline{p}_{k+1} = \underline{p}_k + \beta_d \underline{d}_p$$

end

end

Primal-dual path following algorithm for QP

- QP: $\min_{\underline{x}} \frac{1}{2}\underline{x}^T Q\underline{x} + \underline{d}^T\underline{x} + c \text{ s.t. } A\underline{x} \geq \underline{b}; A\ mxn$

  $KKT: Q\underline{x} - A^T\underline{\lambda} + \underline{d} = \underline{0}; A\underline{x} - \underline{p} - \underline{b} = \underline{0}; p_i\lambda_i = 0; p_i \geq 0; \lambda_i \geq 0$

- Initializ $\underline{p}_0 > 0,\ \underline{\lambda}_0 > 0,\ (\alpha \approx 0.9 - 0.99)$

for $k = 0,1,2,\ldots k_{\max}$

$t = \underline{p}_k^T\underline{\lambda}_k$

If $t < \varepsilon$, stop

else

$\mu_k = t/m$

solve $\begin{bmatrix} Q & -A^T & 0 \\ A & 0 & -I \\ 0 & P_k & \Lambda_k \end{bmatrix}\begin{bmatrix} \underline{d}_{xa} \\ \underline{d}_{\lambda a} \\ \underline{d}_{pa} \end{bmatrix} = -\begin{bmatrix} Q\underline{x}_k - A^T\underline{\lambda}_k + \underline{d} \\ A\underline{x}_k - \underline{p}_k - \underline{b} \\ D_k P_k \underline{e} - \sigma_k\mu_k\underline{e} \end{bmatrix}; \Lambda_k = \text{Diag}(\underline{\lambda}_k); P_k = \text{Diag}(\underline{p}_k)$

calculate

$\beta_a = \min\{\min_{i:d_{\lambda i}<0}(1, -\lambda_{ki}/d_{\lambda ai}), \min_{i:d_{pi}<0}(1, -p_{ki}/d_{pai})\}$

$\mu_{ak} = (\underline{\lambda}_k + \beta_a\underline{d}_{\lambda a})^T(\underline{p}_k + \beta_a\underline{d}_{pa})/m$; Centering parameter $\sigma_k = \left(\mu_{ak}/\mu_k\right)^3$

solve $\begin{bmatrix} Q & -A^T & 0 \\ A & 0 & -I \\ 0 & P_k & \Lambda_k \end{bmatrix}\begin{bmatrix} \underline{d}_{x} \\ \underline{d}_{\lambda} \\ \underline{d}_{p} \end{bmatrix} = -\begin{bmatrix} Q\underline{x}_k - A^T\underline{\lambda}_k + \underline{d} \\ A\underline{x}_k - \underline{p}_k - \underline{b} \\ D_k P_k \underline{e} + \underline{d}_\lambda \circ \underline{d}_p - \sigma_k\mu_k\underline{e} \end{bmatrix}$

$\beta = \{\beta : \underline{p}_k + \beta\underline{d}_p \geq \underline{0}\ \&\ \underline{\lambda}_k + \beta\underline{d}_\lambda \geq \underline{0}\}$

$\underline{x}_{k+1} = \underline{x}_k + \alpha\beta\underline{d}_k; \underline{\lambda}_{k+1} = \underline{\lambda}_k + \alpha\beta\underline{d}_\lambda; \underline{p}_{k+1} = \underline{p}_k + \alpha\beta\underline{d}_p$

end

end

Primal and dual for SOCP

- Recall Cone LP

$$\min_{\underline{x}} \ \underline{c}^T \underline{x}$$
$$\text{s.t.} \quad A\underline{x} = \underline{b}$$
$$\|C_i \underline{x} + \underline{d}_i\|_2 \le \underline{e}_i^T \underline{x} + f_i; i = 1, 2, .., p$$
$$\text{where } \underline{x}, \underline{c}, \underline{e}_i \in R^n; C_i \in R^{k_i - 1 \text{x}\, n}; \underline{d}_i \in R^{k_i - 1}; f_i \in R, A \in R^{m\text{x}n}, \underline{b} \in R^m$$

Explicit form:

$$\min_{\underline{x}} \ \underline{c}^T \underline{x}$$
$$\text{s.t.} \quad A\underline{x} = \underline{b}$$
$$\|\underline{u}_i\|_2 \le t_i; i = 1, 2, .., p$$
$$C_i \underline{x} - \underline{u}_i = -\underline{d}_i; i = 1, 2, .., p$$
$$\underline{e}_i^T \underline{x} - t_i = -f_i; i = 1, 2, .., p$$

- Dual (easy to see from Lagrangian)

$$\max_{\underline{\lambda}, \{\underline{\delta}_i, \gamma_i\}} \ \underline{b}^T \underline{\lambda} - \sum_{i=1}^{p} \left(\underline{d}_i^T \underline{\delta}_i + f_i \gamma_i \right)$$
$$\text{s.t.} \quad \left(\sum_{i=1}^{p} C_i^T \underline{\delta}_i + \underline{e}_i \gamma_i \right) + A^T \underline{\lambda} = \underline{c}$$
$$\|\underline{\delta}_i\|_2 \le \gamma_i; i = 1, 2, .., p$$

Useful in robust SVM and a number of other applications. Interior point methods extend here. See Anderson *et al.*

- KKT or CS conditions:

$$\|\underline{u}_i\|_2 < t_i \Rightarrow \gamma_i = \|\underline{\delta}_i\|_2 = 0$$
$$\|\underline{\delta}_i\|_2 < \gamma_i \Rightarrow t_i = \|\underline{u}_i\|_2 = 0$$
$$\gamma_i = \|\underline{\delta}_i\|_2, \|\underline{u}_i\|_2 = t_i \Rightarrow \gamma_i \underline{u}_i = -t_i \underline{\delta}_i$$

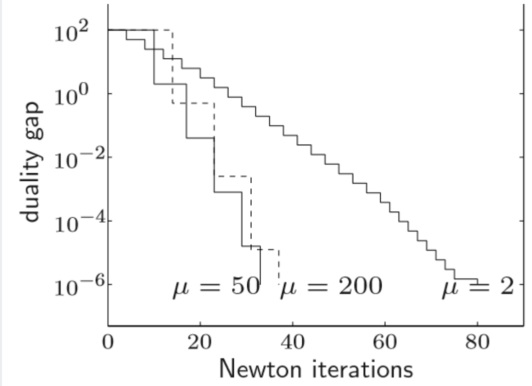
See: Anderson *et al.* “Interior-point methods for large-scale cone programming,” http://www.seas.ucla.edu/~vandenbe/publications/mlbook.pdf
Lobo *et al.* “Applications of second order cone programming,” *Linear Algebra and its Applications*, 284, pp. 193-228, 1998.

Barrier method for SOCP

- Lagrangian of Barrier version of Cone LP

$$\min_{\underline{x}} L(\underline{x},\underline{\lambda}) = \underline{c}^T\underline{x} - \frac{1}{t}\sum_{i=1}^{p} \underbrace{\ln\left[(\underline{e}_i^T\underline{x} + f_i)^2 - \|C_i\underline{x} + \underline{d}_i\|_2^2\right]}_{f_i(\underline{x})} + \underline{\lambda}^T(\underline{b} - A\underline{x})$$

50 variables and 50 cone constraints in R^6



http://www.robots.ox.ac.uk/~az/lectures/b1/vandenberghe_1_2.pdf

- KKT conditions

$$\nabla_{\underline{x}} L = \underline{c} - A^T\underline{\lambda} - \frac{2}{t}\sum_{i=1}^{p}\frac{1}{f_i(\underline{x})}\left[(\underline{e}_i^T\underline{x} + f_i)\underline{e}_i - C_i^T(C_i\underline{x} + \underline{d}_i)\right] = \underline{0}$$

$$\nabla_{\underline{\lambda}} L = -(A\underline{x} - \underline{b}) = \underline{0}$$

- Algorithm: Given a strictly feasible $\underline{x}$ (e.g., phase I LP), $t = t_o \approx 1$, $\mu \approx$ 10-20, tolerance ε
  - Centering step: compute $\underline{x}^*(t)$ and $\underline{\lambda}^*(t)$ set $\underline{x} = \underline{x}^*(t)$ and $\underline{\lambda} = \underline{\lambda}^*(t)$
  - Stopping criterion: Terminate if $p/t < \varepsilon$. Else go to next step.
  - Increase t: $t = \mu t$ and go to Centering step.

- Convergence typically in 20-50 iterations. Primal-dual path following algorithms exist. See Anderson *et al.*

Summary

- Spanning tree algorithms
  - Kruskal
  - Prim
  - Distributed
- Applications to communication network design problem
- Introduction to Cone Programming