Lecture: network flow problems

http://bicmr.pku.edu.cn/~wenzw/bigdata2018.html

Acknowledgement: this slides is based on Prof. James B. Orlin's lecture notes of "15.082/6.855J, Introduction to Network Optimization" at MIT

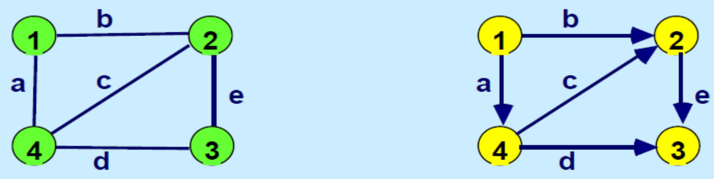
Textbook: Network Flows: Theory, Algorithms, and Applications by Ahuja, Magnanti, and Orlin referred to as AMO

Outline

1. Overview of network flow problems
2. Duality of shortest path problem
3. Duality of Maximum Flows
4. Maximum Bipartite Matching
5. Modularity Maximization for Community Detection

Notation and Terminology

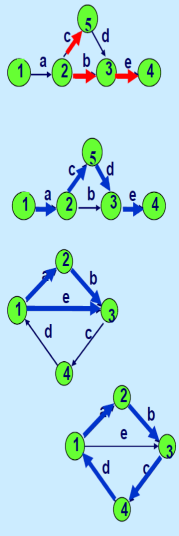
Network terminology as used in AMO.



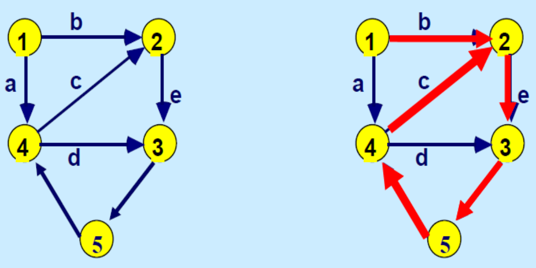
Left: an undirected graph, Right: a directed graph

- Network G = (N, A)
- Node set N = {1, 2, 3, 4}
- Arc set A = {(1,2), (1,3), (3,2), (3,4), (2,4)}
- In an undirected graph, (i,j) = (j,i)

- Path: a finite sequence of nodes: $i_1, i_2, \ldots, i_t$ such that $(i_k, i_{k+1}) \in A$ and all nodes are not the same. Example: 5, 2, 3, 4. (or 5, c, 2, b, 3, e, 4). No node is repeated. Directions are ignored.
- Directed Path. Example: 1, 2, 5, 3, 4 (or 1, a, 2, c, 5, d, 3, e, 4). No node is repeated. Directions are important.
- Cycle (or circuit or loop) 1, 2, 3, 1. (or 1, a, 2, b, 3, e). A path with 2 or more nodes, except that the first node is the last node. Directions are ignored.
- Directed Cycle: (1, 2, 3, 4, 1) or 1, a, 2, b, 3, c, 4, d, 1. No node is repeated. Directions are important.



Walks

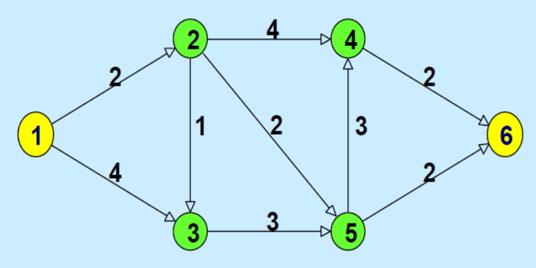


- Walks are paths that can repeat nodes and arcs
- Example of a directed walk: 1-2-3-5-4-2-3-5
- A walk is closed if its first and last nodes are the same.
- A closed walk is a cycle except that it can repeat nodes and arcs.

Three Fundamental Flow Problems

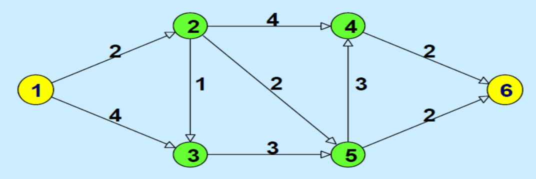
- The shortest path problem
- The maximum flow problem
- The minimum cost flow problem

The shortest path problem



- Consider a network G = (N, A) with cost c_{ij} on each edge $(i, j) \in A$. There is an origin node s and a destination node t.
- Standard notation: n = |N|, m = |A|
- cost of of a path: $c(P) = \sum_{(i,j) \in P} c_{ij}$
- What is the shortest path from s to t?

The shortest path problem



$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t. $\sum_j x_{sj} = 1$

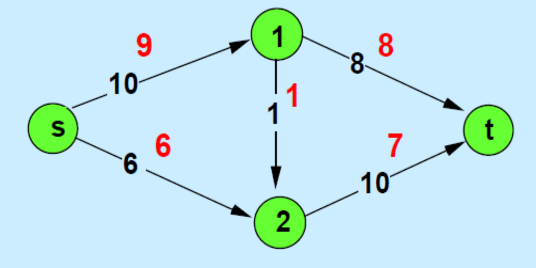
$\sum_j x_{ij} - \sum_j x_{ji} = 0,$ for each i $\neq s$ or t

$-\sum_i x_{it} = -1$

$x_{ij} \in \{0,1\}$ for all (i,j)

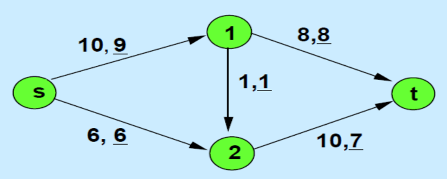
The Maximum Flow Problem

- Directed Graph G = (N, A).
  - Source s
  - Sink t
  - Capacities u_{ij} on arc (i,j)
  - Maximize the flow out of s, subject to
- Flow out of i = Flow into i, for $i \neq s$ or t.



A Network with Arc Capacities (and the maximum flow)

Representing the Max Flow as an LP



Flow out of i = Flow into i, for $i \neq s$ or t.

$$\max \quad v$$

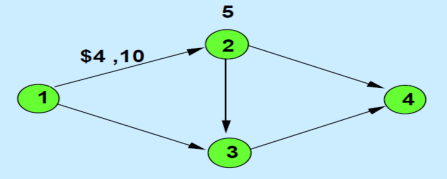
$$\text{s.t.} \quad \sum_j x_{sj} = v$$

$$\sum_j x_{ij} - \sum_j x_{ji} = 0, \quad \text{for each i } \neq s \text{ or } t$$

$$-\sum_i x_{it} = -v$$

$$0 \leq x_{ij} \leq u_{ij} \text{ for all } (i,j)$$

Min Cost Flows



Flow out of i - Flow into i = b(i).
Each arc has a linear cost and a capacity

$$\min \quad \sum_{ij} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j} x_{ij} - \sum_{j} x_{ji} = b(i), \ \text{for each i}$$

$$0 \le x_{ij} \le u_{ij} \ \text{for all} \ (i,j)$$

Covered in detail in Chapter 1 of AMO

Where Network Optimization Arises

- Transportation Systems
  - transportation of goods over transportation networks
  - Scheduling of fleets of airplanes
- Manufacturing Systems
  - Scheduling of goods for manufacturing
  - Flow of manufactured items within inventory systems
- Communication Systems
  - Design and expansion of communication systems
  - Flow of information across networks
- Energy Systems, Financial Systems, and much more

Applications in social network: shortest path

2014 ACM SIGMOD Programming Contest

http://www.cs.albany.edu/~sigmod14contest/task.html

- Shortest Distance Over Frequent Communication Paths
  定义社交网络的边: 相互直接至少有x条回复并且相互认识。给定网络里两个人p1和p2 以及另外一个整数x，寻找图中p1 和p2之间数量最少节点的路径
- Interests with Large Communities
- Socialization Suggestion
- Most Central People (All pairs shorted path)
  定义网络：论坛中有标签t的成员，相互直接认识。给定整数k和标签t,寻找k个有highest closeness centrality values的人

Applications in social network: max flow and etc

Community detection in social network

- Social network is a network of people connected to their “friends”
- Recommending friends is an important practical problem
- solution 1: recommend friends of friends
- solution 2: detect communities
  - idea1: use max-flow min-cut algorithms to find a minimum cut
  - it fails when there are outliers with small degree
  - idea2: find partition A and B that minimize conductance:

$$\min_{A,B} \frac{c(A,B)}{|A|\ |B|},$$

where $c(A,B) = \sum_{i \in A} \sum_{j \in B} c_{ij}$

Outline

1. Overview of network flow problems
2. Duality of shortest path problem
3. Duality of Maximum Flows
4. Maximum Bipartite Matching
5. Modularity Maximization for Community Detection

The shortest path problem: LP relaxation

LP Relaxation: replace $x_{ij} \in \{0, 1\}$ by $x_{ij} \geq 0$

Primal

$$
\begin{aligned}
\min \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\
\text{s.t.} \quad & -\sum_j x_{sj} = -1 \\
& \sum_j x_{ji} - \sum_j x_{ij} = 0, i \neq s \text{ or } t \\
& \sum_i x_{it} = 1 \\
& x_{ij} \geq 0 \text{ for all } (i,j)
\end{aligned}
$$

Dual

$$
\begin{aligned}
\max \quad & d(t) - d(s) \\
\text{s.t.} \quad & d(j) - d(i) \leq c_{ij}, \forall (i,j) \in A
\end{aligned}
$$

Signs in the constraints in the primal problem

Dual LP

Claim: When $G = (N, A)$ satisfies the no-negative-cycles property, the indicator vector of the shortest s-t path is an optimal solution to the LP.

- Let x^* be the indicator vector of shortest s-t path
  - $x^*_{ij} = 1$ if $(i, j) \in P$, otherwise $x^*_{ij} = 0$
  - Feasible for primal
- Let $d^*(v)$ be the shortest path distance from s to v
  - Feasible for dual (by triangle inequality)
- $\sum_{(i,j)\in A} c_{ij}x^*_{ij} = d^*(t) - d^*(s)$
- Hence, both x^* and d^* are optimal

Optimality Conditions

Lemma. Let d*(j) be the shortest path length from node 1 to node j, for each j. Let d( ) be node labels with the following properties:

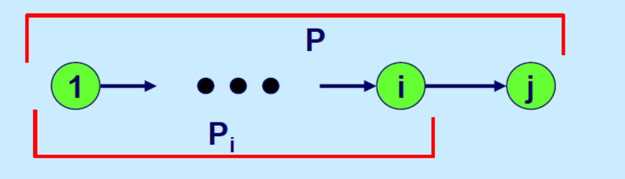
$$d(j) \leq d(i) + c_{ij} \text{ for i } \in \text{ N for j } \neq 1 \quad (1)$$
$$d(1) = 0 \quad (2)$$

Then d(j) $\leq$ d*(j) for each j.

- Proof. Let P be the shortest path from node 1 to node j.

Completion of the proof

- If P = (1, j), then $d(j) \leq d(1) + c_{1j} = c_{1j} = d^*(j)$.
- Suppose |P| > 1, and assume that the result is true for paths of length |P| - 1. Let i be the predecessor of node j on P, and let P_i be the subpath of P from 1 to i.



- P_i is the shortest path from node 1 to node i. So, $d(i) \leq d^*(i) = c(P_i)$by inductive hypothesis. Then, $d(j) \leq d(i) + c_{ij} \leq c(P_i) + c_{ij} = c(P) = d^*(j)$.

Theorem. Let d(1), . . . , d(n) satisfy the following properties for a directed graph G = (N,A):

1. d(1) = 0.
2. d(i) is the length of some path from node 1 to node i.
3. $d(j) \leq d(i) + c_{ij}$ for all (i,j) ∈A.

Then d(j) = d*(j).

Proof. $d(j) \leq d^*(j)$ by the previous lemma. But, $d(j) \geq d^*(j)$ because d(j) is the length of some path from node 1 to node j. Thus d(j) = d*(j).

A Generic Shortest Path Algorithm

Notation.

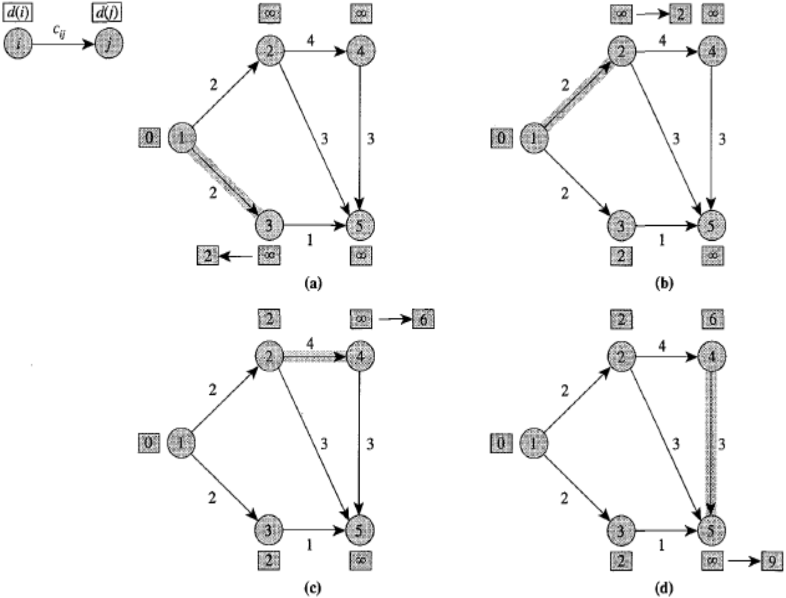
- d(j) = "temporary distance labels".
  - At each iteration, it is the length of a path (or walk) from 1 to j.
  - At the end of the algorithm d(j) is the minimum length of a path from node 1 to node j.
- Pred(j) = Predecessor of j in the path of length d(j) from node 1 to node j.
- c_{ij} = length of arc (i,j).

A Generic Shortest Path Algorithm

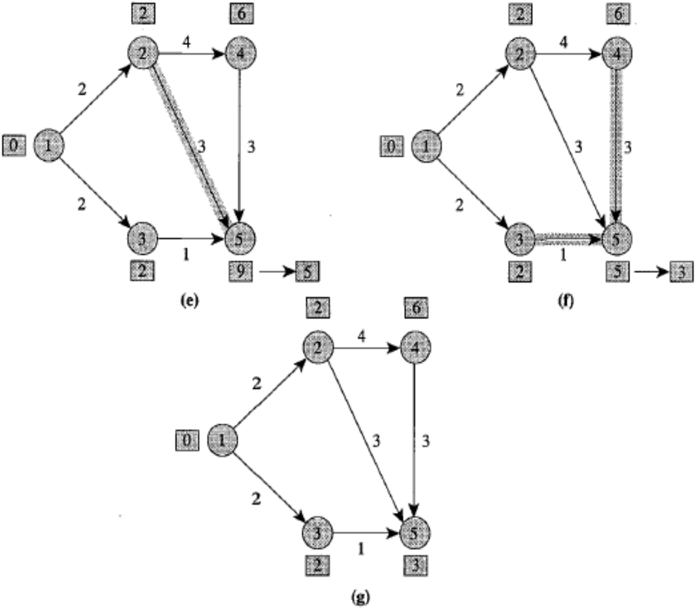
Algorithm LABEL CORRECTING;

- d(1) : = 0 and Pred(1) := $\emptyset$;
  d(j) : = ∞ for each j$\in$N - {1};
- while some arc (i,j) satisfies $d(j) > d(i) + c_{ij}$ do
  $d(j) := d(i) + c_{ij}$;
  Pred(j) : = i;

Ilustration



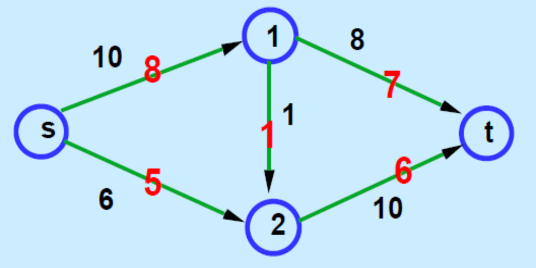
Ilustration



Outline

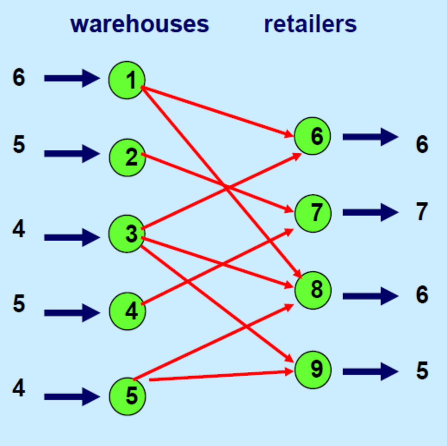
1. Overview of network flow problems
2. Duality of shortest path problem
3. Duality of Maximum Flows
4. Maximum Bipartite Matching
5. Modularity Maximization for Community Detection

We refer to a flow x as maximum if it is feasible and maximizes v. Our objective in the max flow problem is to find a maximum flow.



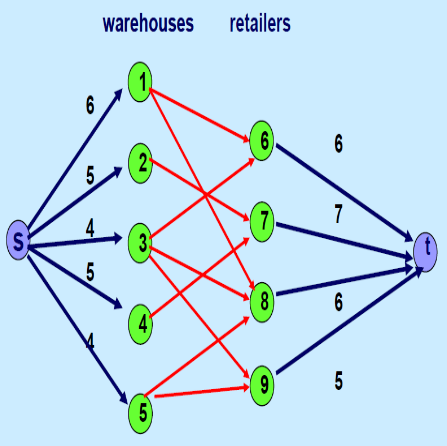
A max flow problem. Capacities and a non- optimum flow.

The feasibility problem: find a feasible flow



Is there a way of shipping from the warehouses to the retailers to satisfy demand?

The feasibility problem: find a feasible flow



There is a 1-1 correspondence with flows from s to t with 24 units (why 24?) and feasible flows for the transportation problem.

The Max Flow Problem

- G = (N,A)
- x_{ij} = flow on arc (i,j)
- u_{ij} = capacity of flow in arc (i,j)
- s = source node
- t = sink node

$$
\begin{aligned}
\max \quad & v \\
\text{s.t.} \quad & \sum_j x_{sj} = v \\
& \sum_j x_{ij} - \sum_j x_{ji} = 0, \ \text{for each i } \neq s \text{ or } t \\
& -\sum_i x_{it} = -v \\
& 0 \leq x_{ij} \leq u_{ij} \text{ for all } (i,j) \in A
\end{aligned}
$$

Dual of the Max Flow Problem

reformulation:

- $A_{i,(i,j)} = 1, A_{j,(i,j)} = -1$, for $(i,j) \in A$ and all other elements are 0
- $A^\top y = y_i - y_j$

The primal-dual pair is

$$\begin{aligned} \min \quad & (\mathbf{0}, -1)(x,v)^\top \\ \text{s.t. } & Ax + (-1, \mathbf{0}, 1)^\top v = 0 \\ & Ix + \mathbf{0}^\top v \le u \\ & x \ge 0, v \text{ is free} \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} \max \quad & -u^\top \pi \\ \text{s.t. } & A^\top y + I^\top \pi \ge 0 \\ & -1 + (-1, \mathbf{0}, 1)y = 0 \\ & \pi \ge 0 \end{aligned}$$

Hence, we have the dual problem:

$$\begin{aligned} \min \quad & u^\top \pi \\ \text{s.t. } & y_j - y_i \le \pi_{ij}, \quad \forall (i,j) \in A \\ & y_t - y_s = 1 \\ & \pi \ge 0 \end{aligned}$$

Duality of the Max Flow Problem

The primal-dual of the max flow problem is

$$
\begin{aligned}
\max \quad & v \\
\text{s.t.} \quad & \sum_j x_{sj} = v \\
& \sum_j x_{ij} - \sum_j x_{ji} = 0, \forall i \notin \{s, t\} \\
& -\sum_i x_{it} = -v \\
& 0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A
\end{aligned}
$$

$$
\begin{aligned}
\min \quad & u^\top \pi \\
\text{s.t.} \quad & y_j - y_i \leq \pi_{ij}, \quad \forall (i,j) \in A \\
& y_t - y_s = 1 \\
& \pi \geq 0
\end{aligned}
$$

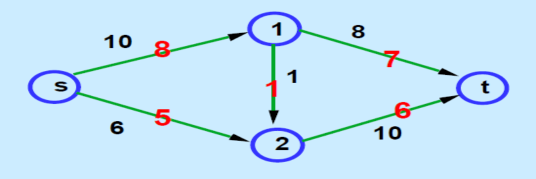
Duality of the Max Flow Problem

- Dual solution describes fraction π_{ij} of each edge to fractionally cut
- Dual constraints require that at least 1 edge is cut on every path P from s to t.

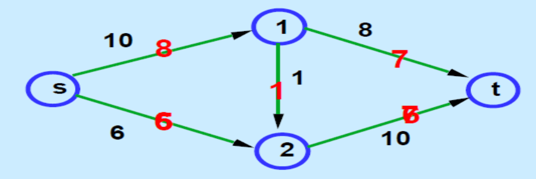
$$\sum_{(i,j)\in P} \pi_{ij} \geq \sum_{(i,j)\in P} y_j - y_i = y_t - y_s = 1$$

- Every integral s-t cut (A,B) is feasible:
  $\pi_{ij} = 1, \forall i \in A, j \in B$, otherwise, $\pi_{ij} = 0$.
  $y_i = 0$ if $i \in A$ and $y_j = 1$ if $i \in B$
- weak duality: $v \leq u^\top \pi$ for any feasible solution
  max flow $\leq$ minimum flow
- strong duality: $v^* = u^\top \pi^*$ at the optimal solution

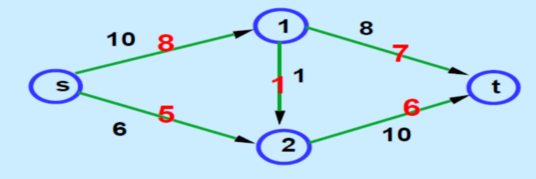
sending flows along s-t paths



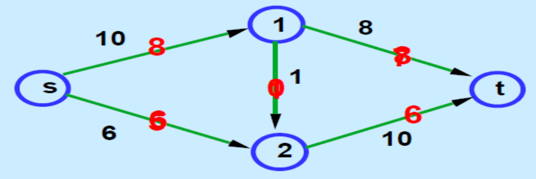
One can find a larger flow from s to t by sending 1 unit of flow along the path s-2-t



A different kind of path

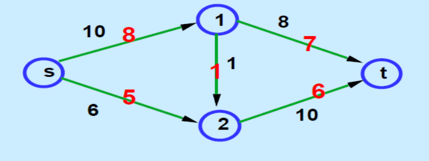


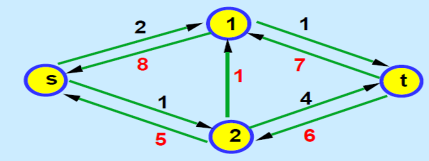
One could also find a larger flow from s to t by sending 1 unit of flow along the path s-2-1-t. (Backward arcs have their flow decreased.)



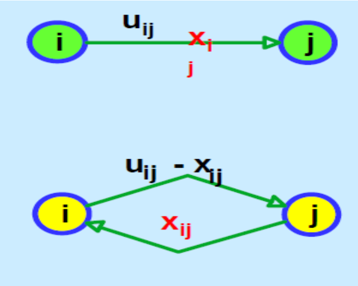
Decreasing flow in (1, 2) is mathematically equivalent to sending flow in (2, 1) w.r.t. node balance constraints.

The Residual Network





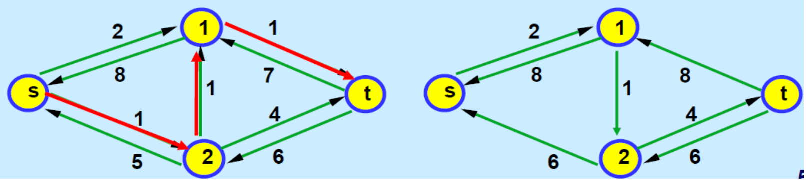
The Residual Network G(x)



We let r_{ij} denote the residual capacity of arc (i,j)

A Useful Idea: Augmenting Paths

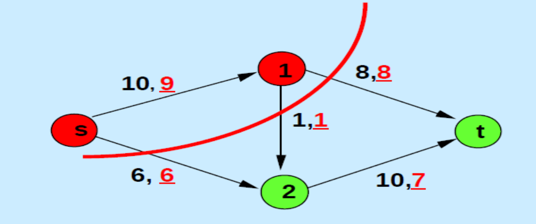
- An augmenting path is a path from s to t in the residual network.
- The residual capacity of the augmenting path P is $\delta(P) = \min\{r_{ij} : (i,j) \in P\}$.
- To augment along P is to send $\delta(P)$ units of flow along each arc of the path. We modify x and the residual capacities appropriately.
- $r_{ij} := r_{ij} - \delta(P)$ and $r_{ji} := r_{ji} + \delta(P)$ for (i,j) $\in$P.



The Ford Fulkerson Maximum Flow Algorithm

- x := 0;
  create the residual network G(x);
- while there is some directed path from s to t in G(x) do
  - let P be a path from s to t in G(x);
  - $\delta := \delta(P) = \min\{r_{ij} : (i,j) \in P\}$;
  - send δ-units of flow along P;
  - update the r's:
    $r_{ij} := r_{ij} - \delta(P)$ and $r_{ji} := r_{ji} + \delta(P)$ for (i,j) $\in$P.

Cut Duality Theory



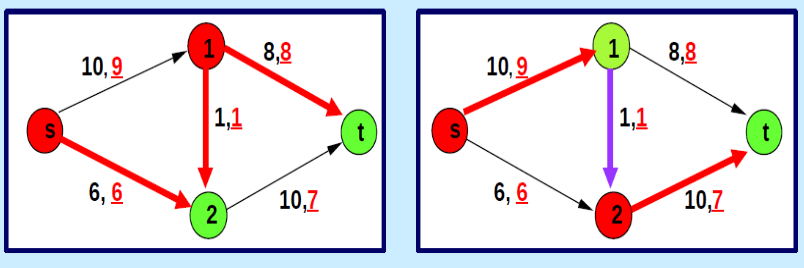
- An (s,t)-cut in a network G = (N,A) is a partition of N into two disjoint subsets S and T such that s $\in$ S and t $\in$T, e.g., S = {s, 1} and T = {2, t}.
- The capacity of a cut (S,T) is

$$\text{cut(S,T)} = \sum_{i \in S} \sum_{j \in T} u_{ij}$$

The flow across a cut

We define the flow across the cut (S,T) to be

$$F_x(S,T) = \sum_{i \in S} \sum_{j \in T} x_{ij} - \sum_{i \in S} \sum_{j \in T} x_{ji}$$



- If S = {s, 1}, then $F_x(S,T)$ = 6 + 1 + 8 = 15
- If S = {s, 2}, then $F_x(S,T)$ = 9 - 1 + 7 = 15

Max Flow Min Cut

Theorem. (Max-flow Min-Cut). The maximum flow value is the minimum value of a cut.

- Proof. The proof will rely on the following three lemmas:
- Lemma 1. For any flow x, and for any s-t cut (S, T), the flow out of s equals $F_x(S, T)$.
- Lemma 2. For any flow x, and for any s-t cut (S, T), $F_x(S, T) \leq \text{cut}(S, T)$.
- Lemma 3. Suppose that x* is a feasible s-t flow with no augmenting path. Let S* = {j : s →j in G(x*)} and let T* = N\S. Then $F_{x^*}(S^*, T^*) = \text{cut}(S^*, T^*)$.

Proof of Theorem (using the 3 lemmas)

- Let x' be a maximum flow
- Let v' be the maximum flow value
- Let x* be the final flow.
- Let v* be the flow out of node s (for x*)
- Let S* be nodes reachable in G(x*) from s.
- Let T* = N$\setminus$S*.

1. v* $\leq$ v', by definition of v'
2. v' = $F_{x'}$ (S*, T*), by Lemma 1.
3. $F_{x'}$ (S*, T*) $\leq$ cut(S*, T*) by Lemma 2.
4. v* = F_{x^*} (S*, T*) = cut(S*, T*) by Lemmas 1,3.

Thus all inequalities are equalities and v* = v'.

Outline

1 Overview of network flow problems

2 Duality of shortest path problem

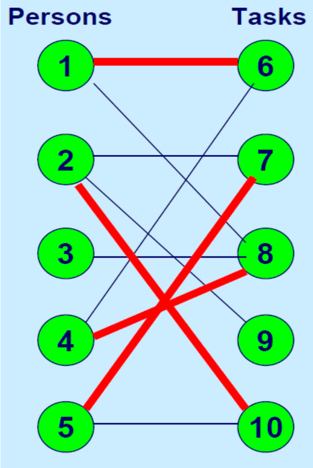
3 Duality of Maximum Flows

4 Maximum Bipartite Matching

5 Modularity Maximization for Community Detection

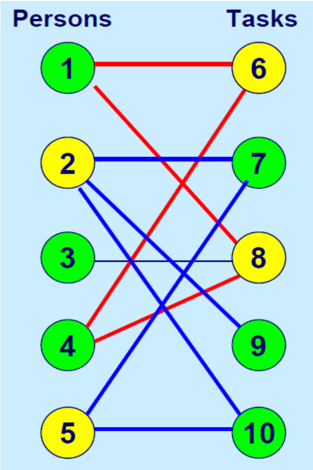
Matchings

- An undirected network G = (N, A) is bipartite if N can be partitioned into N1 and N2 so that for every arc (i,j), i $\in$ N1 and j $\in$ N2.
- A matching in N is a set of arcs no two of which are incident to a common node.
- Matching Problem: Find a matching of maximum cardinality



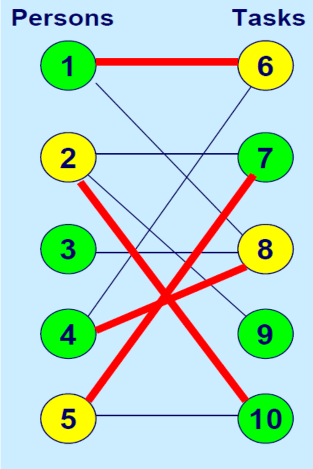
Node Covers

- A node cover is a subset S of nodes such that each arc of G is incident to a node of S.
- Node Cover Problem: Find a node cover of minimum cardinality.

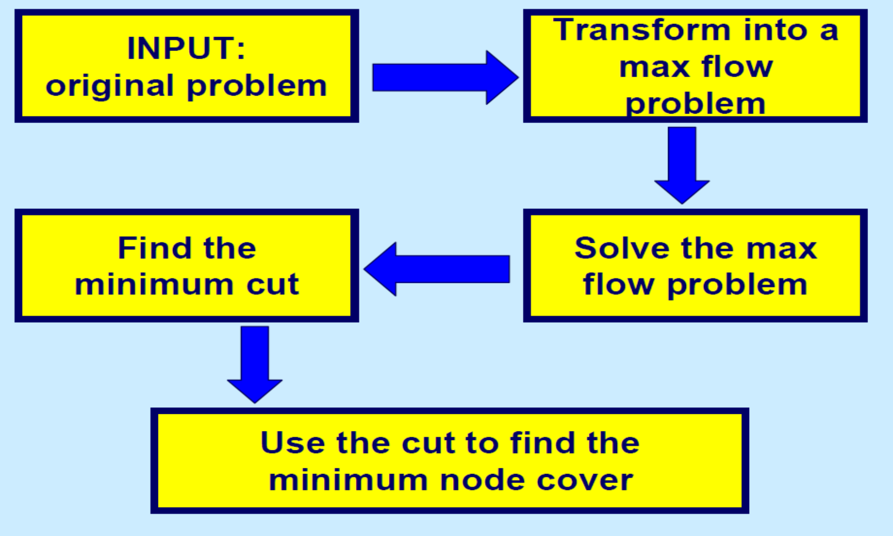


Matching Duality Theorem

- Theorem. König- Egerváry. The maximum cardinality of a matching is equal to the minimum cardinality of a node cover.
- Note. Every node cover has at least as many nodes as any matching because each matched edge is incident to a different node of the node cover.

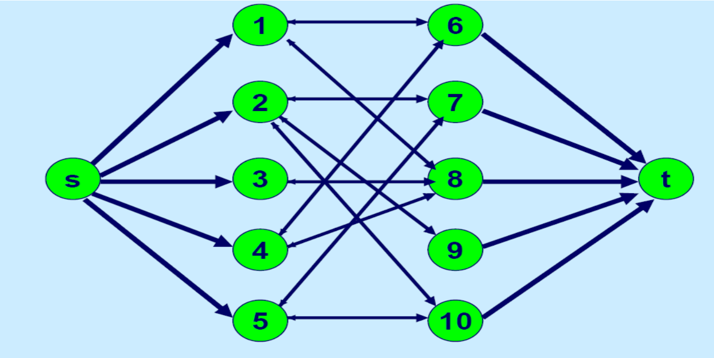


How to find a minimum node cover



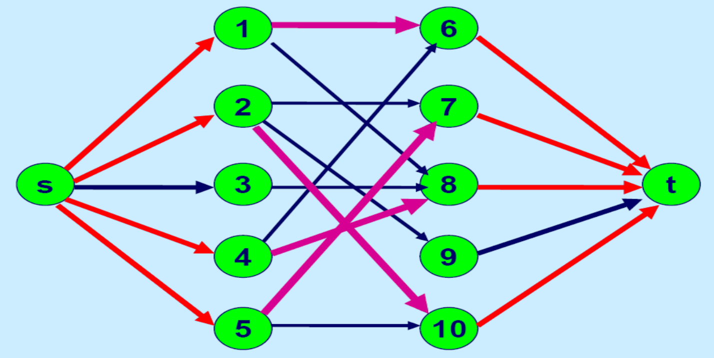
Matching-Max Flow

Solving the Matching Problem as a Max Flow Problem



- Replace original arcs by directed arcs with infinite capacity.
- Each arc (s, i) has a capacity of 1.
- Each arc (j, t) has a capacity of 1.

Find a Max Flow

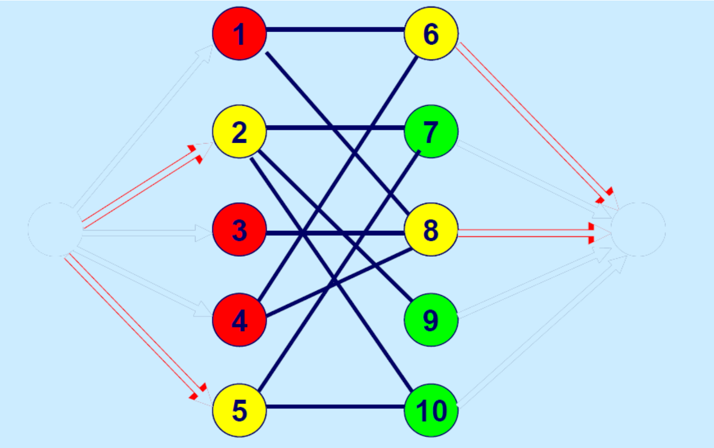


- The maximum s-t flow is 4.
- The max matching has cardinality 4.

Determine the minimum cut

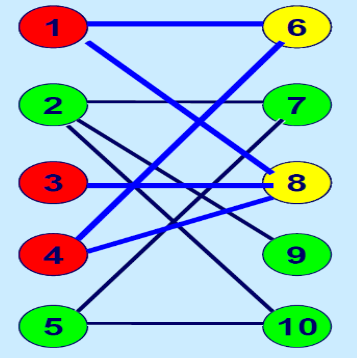
- plot the residual network $G(x)$
- Let S = {j : s $\rightarrow$j in G(x)} and let T = N$\setminus$S.
- S = {s, 1, 3, 4, 6, 8}. T = {2, 5, 7, 9, 10, t}.
- There is no arc from {1, 3, 4} to {7, 9, 10} or from {6, 8} to {2, 5}. Any such arc would have an infinite capacity.

Find the min node cover



- The minimum node cover is the set of nodes incident to the arcs across the cut. Max-Flow Min-Cut implies the duality theorem for matching.
- minimum node cover: {2,5,6,8}

Philip Hall's Theorem

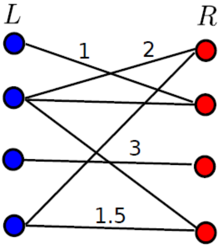


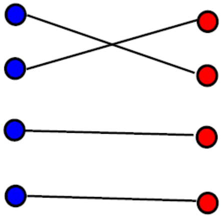
- A perfect matching is a matching which matches all nodes of the graph. That is, every node of the graph is incident to exactly one edge of the matching.
- Philip Hall's Theorem. If there is no perfect matching, then there is a set S of nodes of N1 such that |S| > |T| where T are the nodes of N2 adjacent to S.

The Max-Weight Bipartite Matching Problem

Given a bipartite graph G = (N, A), with N = L ∪ R, and weights w_{ij} on edges (i,j), find a maximum weight matching.

- Matching: a set of edges covering each node at most once
- Let n=|N| and m = |A|.
- Equivalent to maximum weight / minimum cost perfect matching.





The Max-Weight Bipartite Matching

Integer Programming (IP) formulation

$$\max \quad \sum_{ij} w_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j} x_{ij} \leq 1, \forall i \in L$$

$$\sum_{i} x_{ij} \leq 1, \forall j \in R$$

$$x_{ij} \in \{0, 1\}, \forall (i, j) \in A$$

- $x_{ij} = 1$ indicate that we include edge (i, j) in the matching
- IP: non-convex feasible set

The Max-Weight Bipartite Matching

Integer program (IP)

$$\begin{aligned} \max \quad & \sum_{ij} w_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j} x_{ij} \leq 1, \forall i \in L \\ & \sum_{i} x_{ij} \leq 1, \forall j \in R \\ & x_{ij} \in \{0, 1\}, \forall (i,j) \in A \end{aligned}$$

LP relaxation

$$\begin{aligned} \max \quad & \sum_{ij} w_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j} x_{ij} \leq 1, \forall i \in L \\ & \sum_{i} x_{ij} \leq 1, \forall j \in R \\ & x_{ij} \geq 0, \forall (i,j) \in A \end{aligned}$$

- Theorem. The feasible region of the matching LP is the convex hull of indicator vectors of matchings.
- This is the strongest guarantee you could hope for an LP relaxation of a combinatorial problem
- Solving LP is equivalent to solving the combinatorial problem

Primal-Dual Interpretation

Primal LP relaxation

$$\max \quad \sum_{ij} w_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{j} x_{ij} \leq 1, \forall i \in L$$

$$\sum_{i} x_{ij} \leq 1, \forall j \in R$$

$$x_{ij} \geq 0, \forall (i,j) \in A$$

Dual

$$\min \quad \sum_{i} y_i$$

$$\text{s.t.} \quad y_i + y_j \geq w_{ij}, \forall (i,j) \in A$$

$$y \geq 0$$

- Dual problem is solving minimum vertex cover: find smallest set of nodes S such that at least one end of each edge is in S
- From strong duality theorem, we know $P^*_{LP} = D^*_{LP}$

Primal-Dual Interpretation

Suppose edge weights $w_{ij} = 1$, then binary solutions to the dual are node covers.

Dual

$$\min \quad \sum_i y_i$$
$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$
$$y \geq 0$$

Dual Integer Program

$$\min \quad \sum_i y_i$$
$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$
$$y \in \{0, 1\}$$

- Dual problem is solving minimum vertex cover: find smallest set of nodes S such that at least one end of each edge is in S
- From strong duality theorem, we know $P^*_{LP} = D^*_{LP}$
- Consider IP formulation of the dual, then

$$P^*_{IP} \leq P^*_{LP} = D^*_{LP} \leq D^*_{IP}$$

Total Unimodularity

Defintion: A matrix A is Totally Unimodular if every square submatrix has determinant 0, +1 or -1.

Theorem: If $A \in \mathbb{R}^{m \times n}$ is totally unimodular, and b is an integer vector, then $\{x : Ax \leq b; x \geq 0\}$ has integer vertices.

- Non-zero entries of vertex x are solution of $A'x' = b'$ for some nonsignular square submatrix A' and corresponding sub-vector b'
- Cramer's rule:

$$x_i = \frac{\det(A'_i \mid b')}{\det A'}$$

Claim: The constraint matrix of the bipartite matching LP is totally unimodular.

The Minimum weight vertex cover

- undirected graph G = (N, A) with node weights $w_i \geq 0$
- A vertex cover is a set of nodes S such that each edge has at least one end in S
- The weight of a vertex cover is sum of all weights of nodes in the cover
- Find the vertex cover with minimum weight

Integer Program

$$\min \sum_i w_i y_i$$
$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$
$$y \in \{0, 1\}$$

LP Relaxation

$$\min \sum_i w_i y_i$$
$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$
$$y \geq 0$$

LP Relaxation for the Minimum weight vertex cover

- In the LP relaxation, we do not need $y \leq 1$, since the optimal solution y^* of the LP does not change if $y \leq 1$ is added.
  Proof: suppose that there exists an index i such that the optimal solution of the LP y_i^* is strictly larger than one. Then, let y' be a vector which is same as y^* except for $y_i' = 1 < y_i^*$. This y' satisfies all the constraints, and the objective function is smaller.
- The solution of the relaxed LP may not be integer, i.e., $0 < y_i^* < 1$
- rounding technique:

$$y_i' = \begin{cases} 0, & \text{if } y_i^* < 0.5 \\ 1, & \text{if } y_i^* \geq 0.5 \end{cases}$$

- The rounded solution y' is feasible to the original problem

LP Relaxation for the Minimum weight vertex cover

The weight of the vertex cover we get from rounding is at most twice as large as the minimum weight vertex cover.

- Note that $y_i' = \min(\lfloor 2y_i^* \rfloor, 1)$
- Let P_{IP}^* be the optimal solution for IP, and P_{LP}^* be the optimal solution for the LP relaxation
- Since any feasible solution for IP is also feasible in LP, $P_{LP}^* \leq P_{IP}^*$
- The rounded solution y' satisfy

$$\sum_i y_i' w_i \;=\; \sum_i \min(\lfloor 2y_i^* \rfloor, 1) w_i \leq \sum_i 2y_i^* w_i = 2P_{LP}^* \leq 2P_{IP}^*$$

Outline

1. Overview of network flow problems
2. Duality of shortest path problem
3. Duality of Maximum Flows
4. Maximum Bipartite Matching
5. Modularity Maximization for Community Detection

Communities in the Networks

- Many networks have community structures. Nodes in the same cluster have high connection intensity.

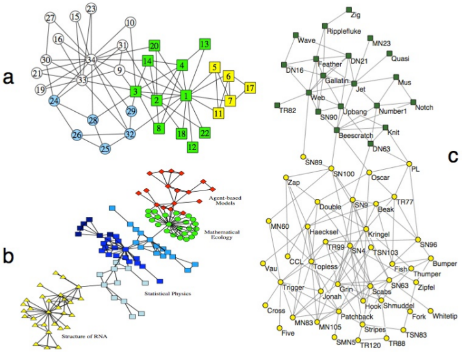


Figure: https://www.slideshare.net/NicolaBarbieri/community-detection

Communities in the Networks

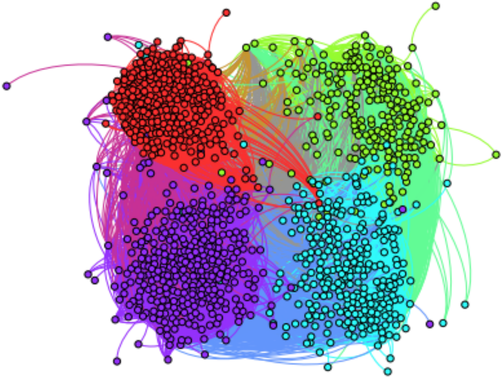


Figure: Simmons College Facebook Network, the four clusters are labeled by different graduation year: 2006 in green, 2007 in light blue, 2008 in purple and 2009 in red. Figure from *Chen, Li and Xu, 2016.*

Partition Matrix and Assignment Matrix

- For any partition $\cup_{a=1}^{k} C_a = [n]$, define the partition matrix X

$$X_{ij} = \begin{cases} 1, & \text{if } i,j \in C_a, \text{ for some } a, \\ 0, & \text{else }. \end{cases}$$

- Low rank solution

$$X = \begin{bmatrix} 1 & & & & & \\ & 1 & 1 & 1 & & \\ & 1 & 1 & 1 & & \\ & 1 & 1 & 1 & & \\ & & & & 1 & 1 \\ & & & & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ & 1 & \\ & 1 & \\ & 1 & \\ & & 1 \\ & & 1 \end{bmatrix} \times \begin{bmatrix} 1 & & & & & \\ & 1 & 1 & 1 & & \\ & & & & 1 & 1 \end{bmatrix}$$

Modularity Maximization

- The modularity (*MEJ Newman, M Girvan, 2004*) is defined by

$$Q = \langle A - \frac{1}{2\lambda} dd^T, X \rangle$$

  where $\lambda = |E|$.

- The Integral modularity maximization problem:

$$\begin{array}{rl} \max & \langle A - \frac{1}{2\lambda} dd^T, X \rangle \\ \text{s.t.} & X \in \{0,1\}^{n \times n} \text{ is a partiton matrix.} \end{array}$$

- Probably hard to solve.

Modularity Maximization: SDP relaxation

- The modularity (*MEJ Newman, M Girvan, 2004*) is defined by

$$Q = \langle A - \frac{1}{2\lambda} dd^T, X \rangle$$

  where $\lambda = |E|$.

- SDP Relaxation Yudong Chen, Xiaodong Li, Jiaming Xu

$$\begin{array}{rl} \max & \langle A - \frac{1}{2\lambda} dd^T, X \rangle \\ \text{s.t.} & X \succeq 0 \\ & 0 \le X_{ij} \le 1 \\ & X_{ii} = 1 \end{array}$$

A Nonconvex Completely Positive Relaxation

- A nonconvex completely positive relaxation of modularity maximization:

$$\begin{aligned} &\min \langle -A + \frac{1}{2\lambda} dd^T, UU^T \rangle \\ &\text{s.t.} U \in \mathbb{R}^{n \times k} \\ &\quad \|u_i\|^2 = 1, \|u_i\|_0 \leq p, i = 1, \ldots, n, \\ &\quad U \geq 0 \end{aligned}$$

- $\|u_i\|^2 = 1$: helpful in the algorithm.
- $U \geq 0$: important in theoretical proof.
- $\|u_i\|_0 \leq p$: keep the sparsity.

A Nonconvex Proximal RBR Algorithm

- Define

$$\mathcal{U}_i := \{u_i \in \mathbb{R}^k \mid u_i \geq 0, \|u_i\|_2 = 1, \|u_i\|_0 \leq p\}$$

- Define

$$\mathcal{U} := \mathcal{U}_1 \times \ldots \times \mathcal{U}_n$$

then rewrite U in component-wise form:

$$U = [u_1, u_2, \ldots, u_n]^T$$

- Rewrite the problem as

$$\min_{U \in \mathcal{U}} f(U) \equiv \langle C, UU^T \rangle$$

A Nonconvex Proximal RBR Algorithm

- Proximal BCD reformulation: fix the other rows and minimize over the ith row

$$u_i = \operatorname*{argmin}_{x \in \mathcal{U}_i} f(u_1, \ldots, u_{i-1}, x, u_{i+1}, \ldots, u_n) + \frac{\sigma}{2} \|x - \bar{u}_i\|^2$$

- Work in blocks:

$$C = \begin{bmatrix} C_{11} & C_{1i} & C_{1n} \\ C_{i1} & c_{ii} & C_{in} \\ C_{n1} & C_{ni} & C_{nn} \end{bmatrix}, \quad UU^T = \begin{bmatrix} U_1^T U_1 & U_1^T x & U_1^T U_n \\ x^T U_1 & x^T x & x^T U_n \\ U_n^T U_1 & U_n^T x & U_n^T U_n \end{bmatrix}$$

- Note that $\|x\| = 1$. The problem is simplified to

$$u_i = \operatorname*{argmin}_{x \in \mathcal{U}_i} b^T x,$$

where

$$b^T = 2C^i_{-i} U_{-i} - \sigma \bar{u}_i^T.$$

Randomized BCD Algorithm

Algorithm 1: Low-rank Decomposition Row by Row (RBR) method

Give U^0, set $k = 0$

while *Not converging* **do**

$u_{i_1}^{k+1} = \arg\min_{x \in \mathcal{U}_{i_1}} f(x, u_{i_2}^k, \ldots, u_{i_n}^k) + \frac{\sigma}{2} \|x - u_{i_1}^k\|^2$

$\vdots$

$u_{i_n}^{k+1} = \arg\min_{x \in \mathcal{U}_{i_n}} f(u_{i_1}^{k+1}, \ldots, u_{i_{n-1}}^{k+1}, x) + \frac{\sigma}{2} \|x - u_{i_n}^k\|^2$

Extract the community by k-means or direct rounding from U^*.

- $\mathcal{U}_i = \{u_i \in \mathbb{R}^k \mid \|u_i\|_2 = 1, u_i \geq 0, \|u_i\|_0 \leq p\}, \mathcal{U} = \mathcal{U}_1 \times \cdots \times \mathcal{U}_n$.
- Each sub-problem: $u_i = \arg\min_{x \in \mathcal{U}_i} b^\top x$ Explicit solution

$$u = \begin{cases} \frac{b_p^-}{\|b_p^-\|}, & \text{if } b^- \neq 0, \\ e_{j_0}, \text{ with } j_0 = \arg\min_j b_j, & \text{otherwise.} \end{cases}$$

Complexity and Implementation Issues

- Expand the matrix C to get b^T:

$$b^T = -2A^i_{-i}U_{-i} + 2\lambda d_i d^T_{-i}U_{-i} - \sigma \bar{u_i}^T$$

- Compute $-A^i_{-i}U_{-i}$: $\mathcal{O}(d_i p)$ FLOPS.
- Compute $d_i d^T_{-i}U_{-i}$ using

$$d^T U = d^T_{-i}U_{-i} + d_i u_i^T$$

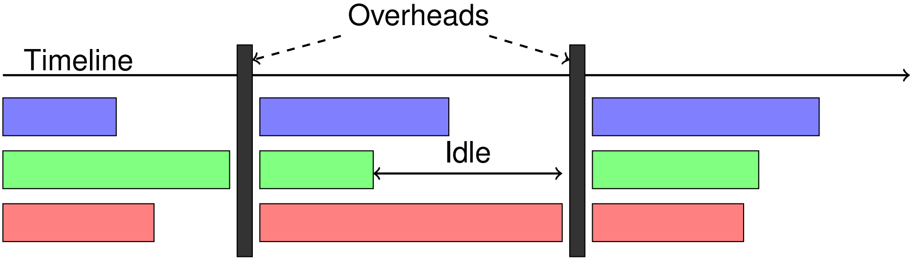
- Update $d^T U$ using

$$d^T U \leftarrow d^T U + d_i(u_i^T - \bar{u_i}^T)$$

Asynchronous Updates

Q: How to deal with the conflicts?
A: Asynchronous programming tells us to just ignore it.

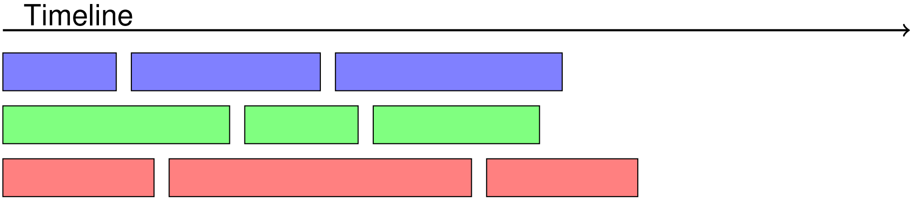
The synchronous world:



- Load imbalance causes the idle.
- Correct but slow.

Asynchronous Updates

The asynchronous world:



- No synchronizations among the workers.
- No idle time – every worker is kept busy.
- High scalability.
- Noisy but fast.

An Asynchronous Proximal RBR Algorithm

Algorithm 2: Asynchronous parallel RBR algorithm

Give U^0, set $t = 0$
while *Not converging* **do**
    for *each row i asynchronously* **do**
        Compute the vector $b_i^\top = -2A^i_{-i}U_{-i} + 2\lambda d_i d_{-i}^\top U_{-i} - \sigma u_i$, and save previous iterate $\bar{u}_i$ in the private memory.
        Update $u_i \leftarrow \mathrm{argmin}_{x \in \mathcal{U}_i} b_i^\top x$ in the shared memory.
        Update the vector $d^\top U \leftarrow d^\top U + d_i(u_i - \bar{u}_i)$ in the shared memory.
    if *rounding is activated* **then**
        for *each row i asynchronously* **do**
            Set $u_i = e_{j_0}$ where $j_0 = \arg\max(u_i)_j$.
        Compute and update $d^\top U$.