CS5340
Uncertainty Modeling in AI

Lecture 12:
Graph-Cut and Alpha-Expansion

Asst. Prof. Lee Gim Hee

AY 2018/19

Semester 1

Course Schedule

Week	Date	Topic	Remarks
1	15 Aug	Introduction to probabilities and probability distributions	
2	22 Aug	Fitting probability models	Hari Raya Haji*
3	29 Aug	Bayesian networks (Directed graphical models)	
4	05 Sep	Markov random Fields (Undirected graphical models)	
5	12 Sep	I will be traveling	No Lecture
6	19 Sep	Variable elimination and belief propagation	
-	26 Sep	Recess week	No lecture
7	03 Oct	Factor graph and the junction tree algorithm	
8	10 Oct	Parameter learning with complete data	
9	17 Oct	Mixture models and the EM algorithm	
10	24 Oct	Hidden Markov Models (HMM)	
11	31 Oct	Monte Carlo inference (Sampling)	
12	07 Nov	Variational inference	
13	14 Nov	Graph-cut and alpha expansion	

* **Make-up lecture**: 25 Aug (Sat), 9.30am-12.30pm, LT 15

Acknowledgements

- A lot of slides and content of this lecture are adopted from:

1. "Computer vision: models, learning and inference", Simon J.D. Prince, Chapter 12

2. http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

3. "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision", PAMI 2004

Learning Outcomes

- Students should be able to:

1. Convert the binary/multi- labeling problem into a max-flow/min-cut problem.

2. Solve the max-flow/min-cut problem using the augmented path algorithm.

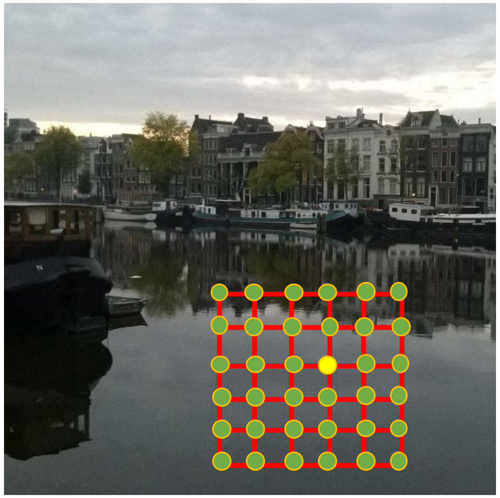
3. Explain the concept of sub-modularity.

4. Solve submodular binary/multi- labeling problem with max-flow/min-cut and non-submodular multi-labelling problem with alpha-expansion.

Motivation

- Consider a grid structure Markov Random Field, e.g. one unknown world state at each pixel in an image.
- Loops in the model, hence, exact inference, i.e. belief propagation cannot be used.
- Can we do better than approximate inference?

Photo Source:
G.H. Lee “Amsterdam”, Oct’16



: pixel is labeled as “water”

: is this pixel more likely to be “water” or “sky”?

Motivation

- **Question:** Can we do better than approximate inference?

- **Answer:** Yes, MRF inference using a set of techniques called graph-cuts!

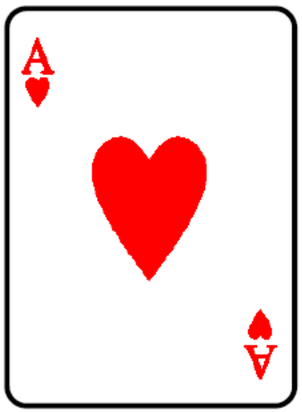
- Three cases of Graph-cuts:

MRF Type	Costs	Inference
Binary-label	Submodular	Exact
Multi-label	Submodular	Exact
Multi-label	Non-Submodular	Approximate (some cases)

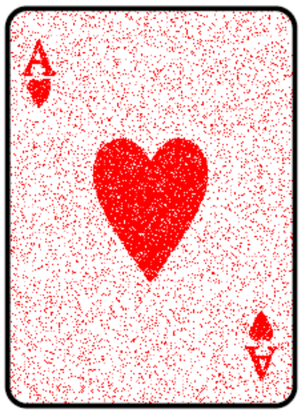
Image Denoising

- We will use image denoising example to illustrate the binary- and multi- labeling tasks.

Binary-Label



Before



After

- Image represented as binary discrete variables
- Proportion of pixels randomly changed polarity

Multi-Label

Before

After

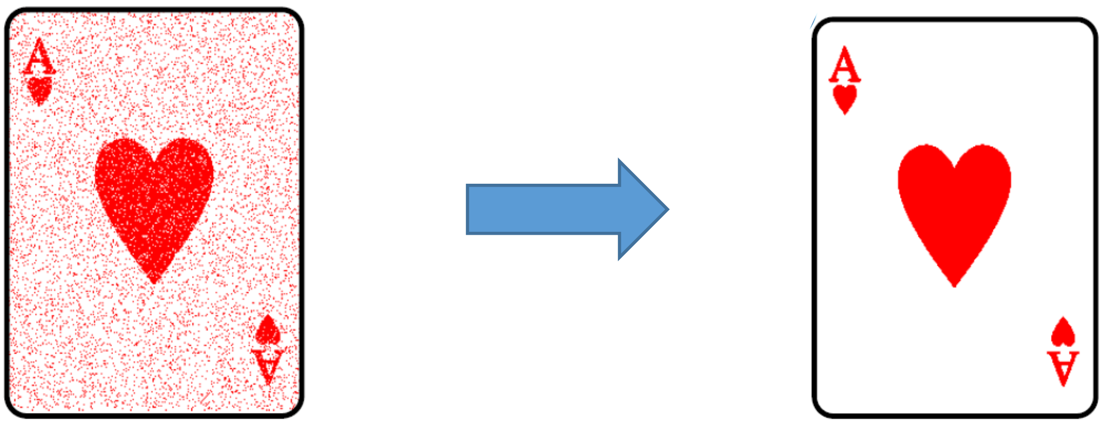
- Image represented as discrete variables representing intensity
- Proportion of pixels randomly changed according to a uniform distribution

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Image Denoising Goal

- **Goal:**

  To recover the clean image pixels, $\boldsymbol{W} = \{W_1, W_2, \dots, W_N\}$, from the given noisy observed data, $\boldsymbol{X} = \{X_1, X_2, \dots, X_N\}$.



Given: Observed Data

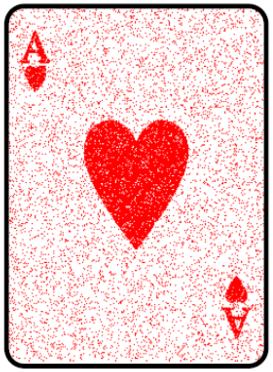
$\boldsymbol{X} = \{X_1, X_2, \dots, X_N\}$

Recovered: Uncorrupted Image

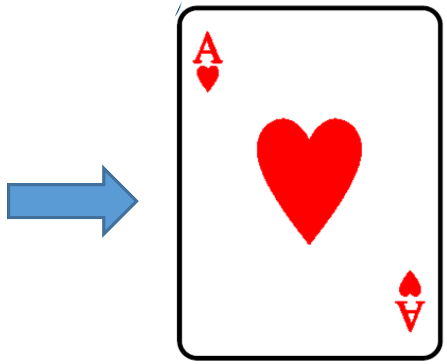
$\boldsymbol{W} = \{W_1, W_2, \dots, W_N\}$

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

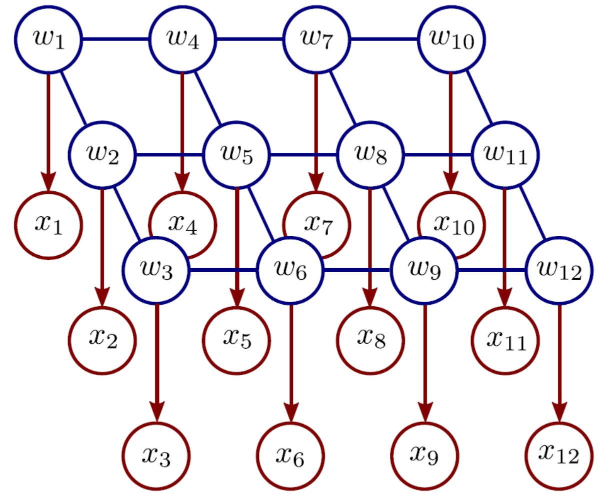
Markov Random Fields



Given: Observed Data
$\boldsymbol{X} = \{X_1, X_2, \dots, X_N\}$



Recovered: Uncorrupted Image
$\boldsymbol{W} = \{W_1, W_2, \dots, W_N\}$



Main idea:

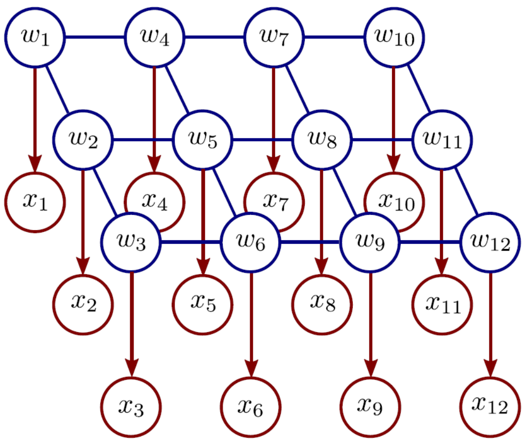
We use a MRF model that encourages the pixels to:

1. stay the same as the observation (likelihood), and
2. take the same label as its neighbors, i.e. pairwise smoothness (prior).

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Markov Random Fields

- More formally, the MRF is given by:



$$p(w_{1\ldots N}|x_{1\ldots N}) = \frac{\prod_{n=1}^{N} p(x_n|w_n)\ p(w_{1\ldots N})}{p(x_{1\ldots N})},$$

(Bayes' rule)

where

- MRF Prior (pairwise potentials): $p(w_{1\ldots N}) = \frac{1}{Z}\exp\left[-\sum_{(m,n)\in\mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}]\right]$

- Likelihoods (unary potentials):

$$\begin{aligned} p(x_n|w_n = 0) &= \text{Bern}_{x_n}[\rho] \\ p(x_n|w_n = 1) &= \underbrace{\text{Bern}_{x_n}[1-\rho]}_{\text{Bernoulli Distribution}} \end{aligned}$$

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

MAP Inference

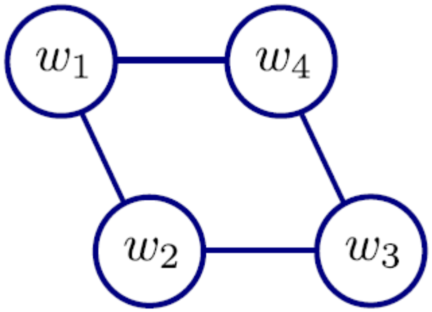
$$
\begin{aligned}
\hat{w}_{1\ldots N} &= \underset{w_{1\ldots N}}{\operatorname{argmax}} \left[\; p\left(w_{1\ldots N} | \mathbf{x}_{1\ldots N}\right)\right] \\
&= \underset{w_{1\ldots N}}{\operatorname{argmax}} \left[\prod_{n=1}^{N} \; p\left(x_n | w_n\right) \; p\left(w_{1\ldots N}\right)\right] && \text{(Bayes' rule)} \\
&= \underset{w_{1\ldots N}}{\operatorname{argmax}} \left[\sum_{n=1}^{N} \log\left[\; p\left(x_n | w_n\right)\right] + \log\left[\; p\left(w_{1\ldots N}\right)\right]\right] && \text{(Log posterior)} \\
&= \underset{w_{1\ldots N}}{\operatorname{argmax}} \left[\sum_{n=1}^{N} \log\left[\; p(x_n | w_n)\right] - \sum_{(m,n)\in\mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}]\right] \\
&= \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[\sum_{n=1}^{N} -\log\left[\; p\left(x_n | w_n\right)\right] + \sum_{(m,n)\in\mathcal{C}} \psi[w_m, w_n, \boldsymbol{\theta}]\right] && \text{(-ve Log posterior)} \\
&= \underset{w_{1\ldots N}}{\operatorname{argmin}} \left[\boxed{\sum_{n=1}^{N} U_n(w_n)} + \boxed{\sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n)}\right],
\end{aligned}
$$

Unary terms
(compatibility of data with label *w*)

Pairwise terms
(compatibility of neighboring labels)

Pairwise Smoothness

Example: Consider a 4-node MRF with only pairwise smoothness potentials (no observations).



$$p(\mathbf{w}) = \frac{1}{Z}\phi_{12}(w_1, w_2)\phi_{23}(w_2, w_3)\phi_{34}(w_3, w_4)\phi_{41}(w_4, w_1)$$

$$\phi_{mn}(0,0) = 1.0 \qquad \phi_{mn}(0,1) = 0.1$$
$$\phi_{mn}(1,0) = 0.1 \qquad \phi_{mn}(1,1) = 1.0$$

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Pairwise Smoothness

$$p(\mathbf{w}) = \frac{1}{Z}\phi_{12}(w_1, w_2)\phi_{23}(w_2, w_3)\phi_{34}(w_3, w_4)\phi_{41}(w_4, w_1)$$

$$\phi_{mn}(0,0) = 1.0 \qquad \phi_{mn}(0,1) = 0.1$$

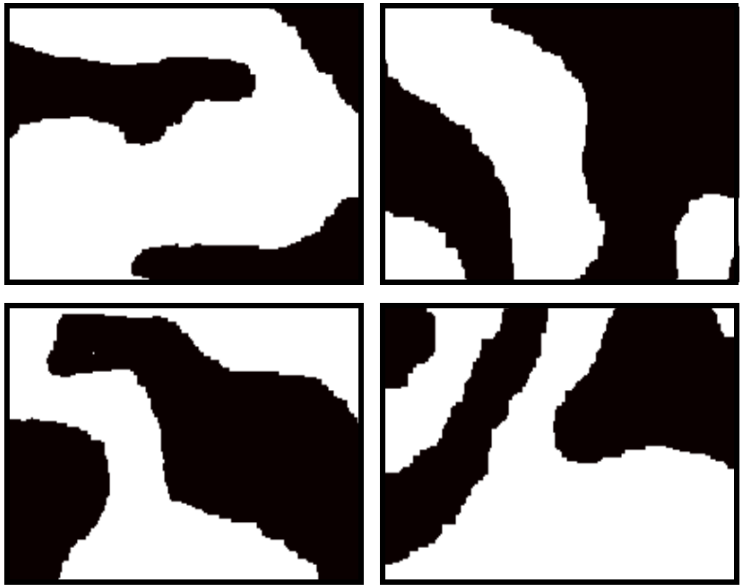
$$\phi_{mn}(1,0) = 0.1 \qquad \phi_{mn}(1,1) = 1.0$$

Probability table:

$w_{1\ldots4}$	$Pr(w_{1\ldots4})$	$w_{1\ldots4}$	$Pr(w_{1\ldots4})$	$w_{1\ldots4}$	$Pr(w_{1\ldots4})$	$w_{1\ldots4}$	$Pr(w_{1\ldots4})$
0000	0.47176	0100	0.00471	1000	0.00471	1100	0.00471
0001	0.00471	0101	0.00005	1001	0.00471	1101	0.00471
0010	0.00471	0110	0.00471	1010	0.00005	1110	0.00471
0011	0.00471	0111	0.00471	1011	0.00471	1111	0.47176

- Smooth solutions (e.g. 0000,1111) have high probability
- Z computed by summing the 16 un-normalized probabilities

Pairwise Smoothness



- Figure shows samples from larger grid which are mostly smooth (higher probability of getting sampled).
- Sampling is used because it is intractable to compute the partition function Z

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Graph-Cuts Overview

- Graph-cuts are used to optimize this cost function:

$$\underset{w_{1\ldots N}}{\arg\min} \boxed{\sum_{n=1}^{N} U_n(w_n)} + \boxed{\sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n)},$$

Unary terms
(compatibility of data with label w)

Pairwise terms
(compatibility of neighboring labels)

- Three cases of Graph-cuts:

MRF Type	Costs	Inference
Binary-label, $w_i \in \{0,1\}$	Submodular	Exact
Multi-label, $w_i \in \{1, \ldots, K\}$	Submodular	Exact
Multi-label, $w_i \in \{1, \ldots, K\}$	Non-Submodular	Approximate (some cases)

Graph-Cuts Overview

- Graph-cuts are used to optimize this cost function:

$$\arg\min_{w_{1\ldots N}} \boxed{\sum_{n=1}^{N} U_n(w_n)} + \boxed{\sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n)},$$

Unary terms
(compatibility of data with label w)

Pairwise terms
(compatibility of neighboring labels)

Approach:

- Convert minimization into a graph problem, i.e.

MAXIMUM FLOW or MINIMUM CUT ON A GRAPH!

- Polynomial-time methods for solving this problem are known

Max-Flow Min-Cut Problem

- "Free world goal": minimum cuts (to the railway tracks) to stop the flow of supplies from St. Petersburg (Source node, S) to Moscow (Sink node, T)?

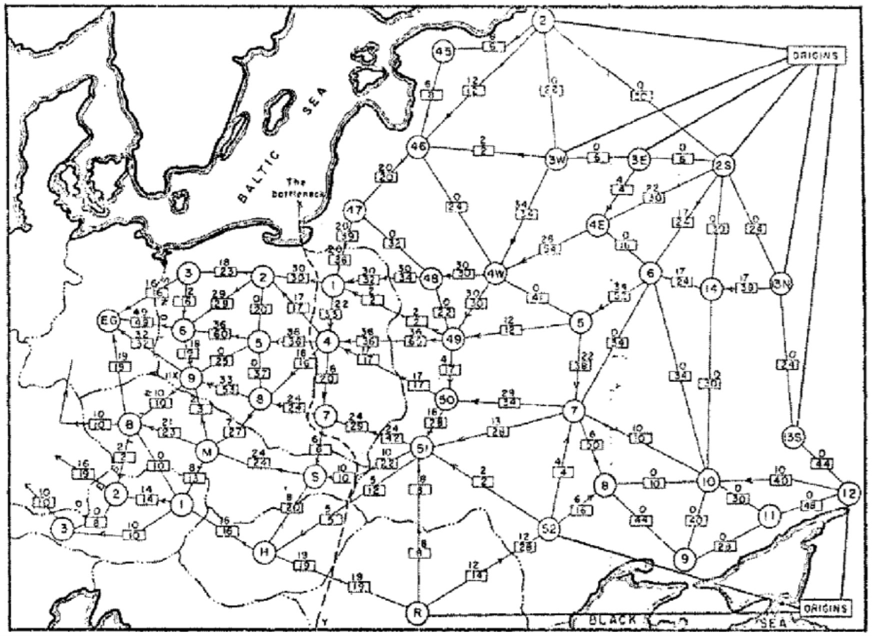


Image source: On the history of the transportation and maximum flow problems. Alexander Schrijver in Math Programming, 91: 3, 2002.

Max-Flow Min-Cut: Definitions

- Given a directed graph, $\mathcal{G}(E, V)$, where $E = \{\ldots, e_{ij}, \ldots\}$ and $V = \{v_1, \ldots, v_i, \ldots\}$ represent the directed edges and nodes.
- Let $s \in V$ be the source and $t \in V$ be the sink of $\mathcal{G}$, and denote the flow through an edge by $f(e_{ij})$.
- The capacity, $u(e_{ij}) \geq 0$, of an edge is the maximum amount of flow that can pass through an edge.

Example:

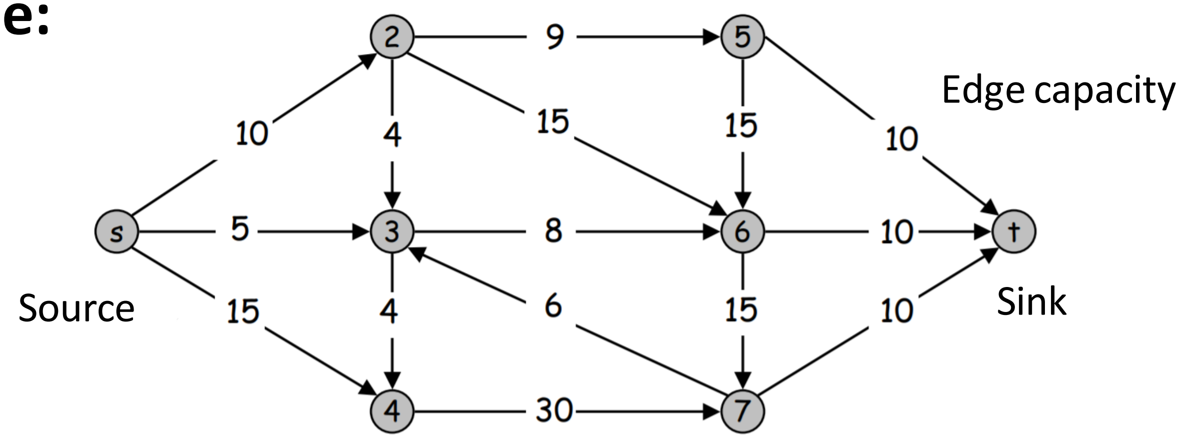


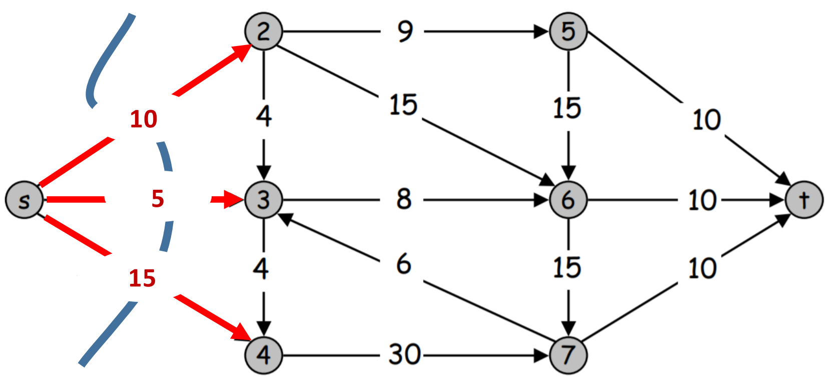
Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

What is a Cut?

- A cut is a node partition (S, T) on $\mathcal{G}$ such that $s \in S$ and $t \in T$, where $S \cap T = \emptyset$ and $S \cup T = V$.
- Capacity(S, T) = sum of weights of edges **leaving** S.

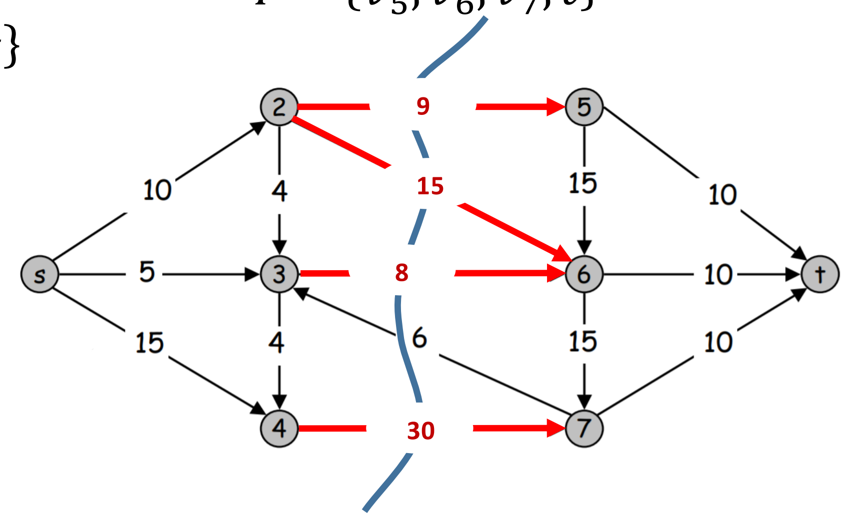
Examples:

$S = \{s\}$
$T = \{v_2, v_3, v_4, v_5, v_6, v_7, t\}$



Capacity$(S,T) = 30$

$S = \{s, v_2, v_3, v_4\}$
$T = \{v_5, v_6, v_7, t\}$



Capacity$(S,T) = 62$

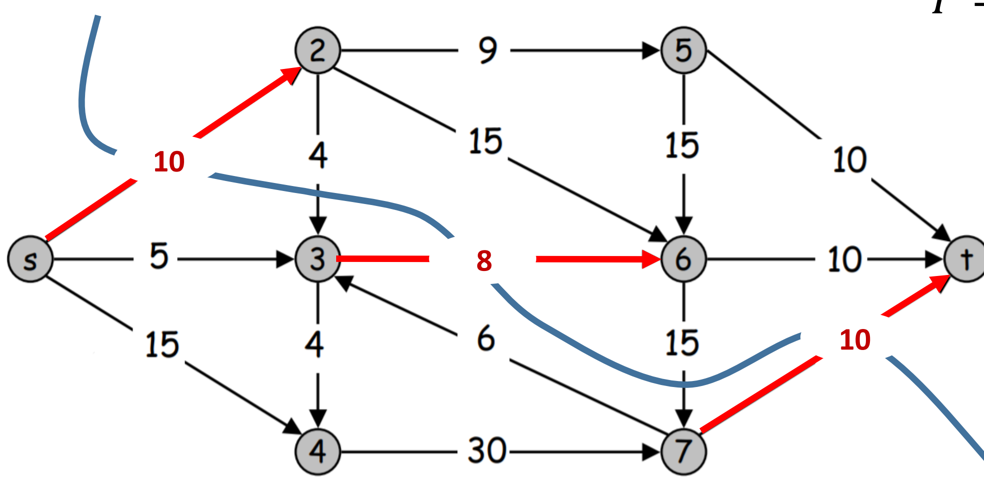
Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

Min-Cut Problem

- Find an s-t cut of minimum capacity!

Example:

$S = \{s, v_3, v_4, v_7\}$
$T = \{v_2, v_5, v_6, t\}$



$\text{Capacity}(S, T) = 28$

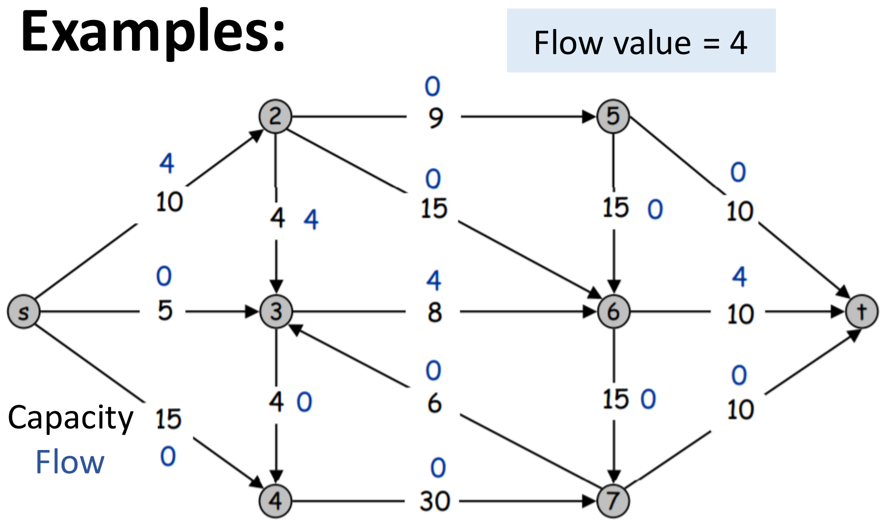
Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

What is a Flow?

- A flow f is an assignment of weights to edges so that:

1. Capacity: $0 \leq f(e_{ij}) \leq u(e_{ij})$.
2. Flow conservation, i.e., flow leaving v_i = flow entering v_i (except at s and t).

Examples:



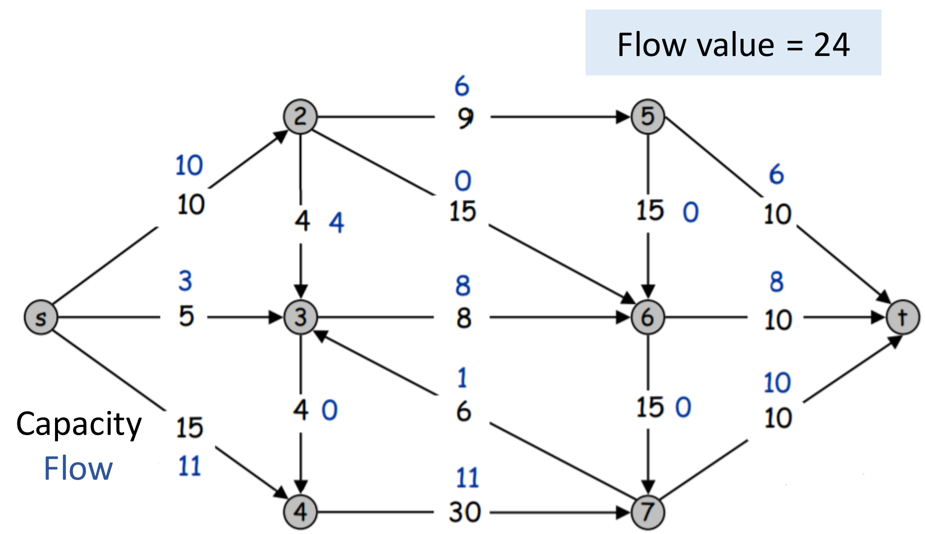


Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

Max-Flow Problem

- Find the flow that maximizes the net flow into the sink.

Example:

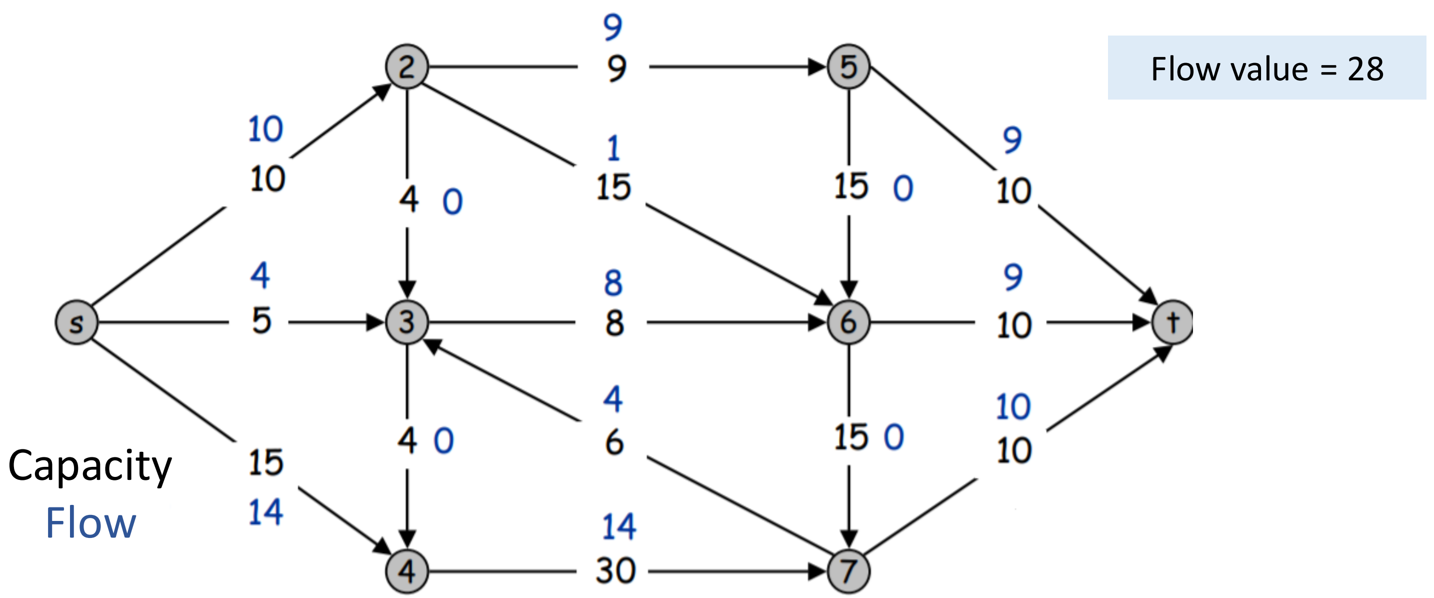


Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

Flows and Cuts

- **Observation 1**: Let f be a flow, and let (S, T) be any s-t cut. Then, the net flow sent across the cut is equal to the amount reaching t.

Example:

$S = \{s, v_2, v_3, v_4\}$
$T = \{v_5, v_6, v_7, t\}$

Flow value = 24

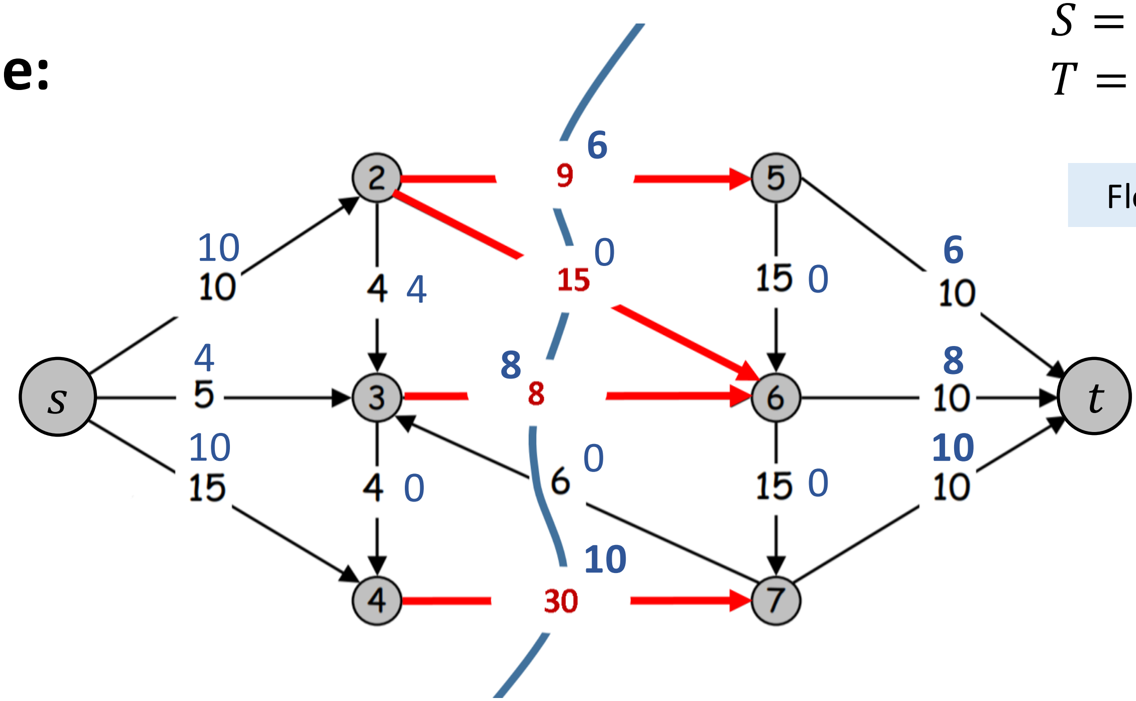


Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

Flows and Cuts

- **Observation 2:** Let f be a flow, and let (S, T) be any s-t cut. Then, the value of the flow is at most the capacity of the cut.

Example:

Cut Capacity = 30 $\Rightarrow$ Flow value ≤ 30

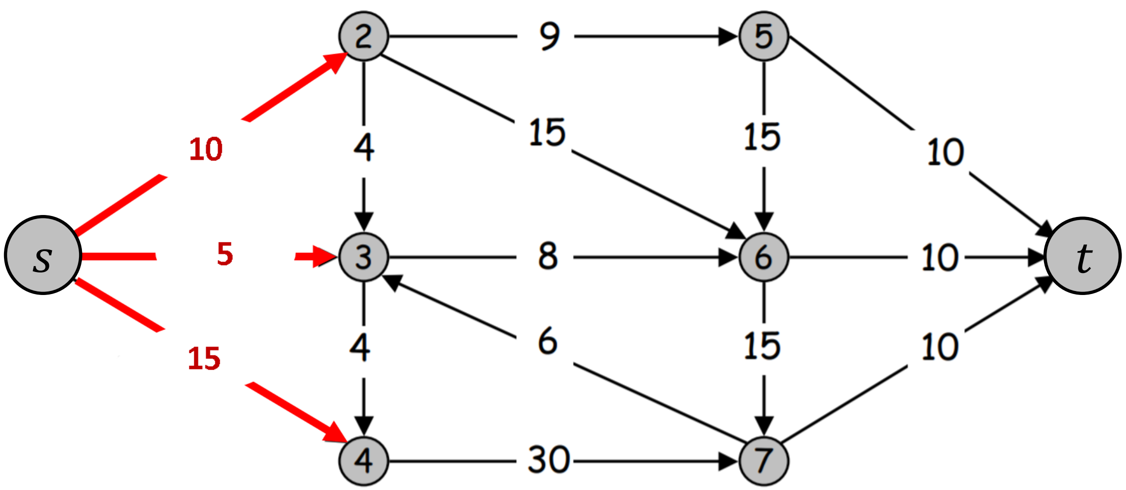


Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

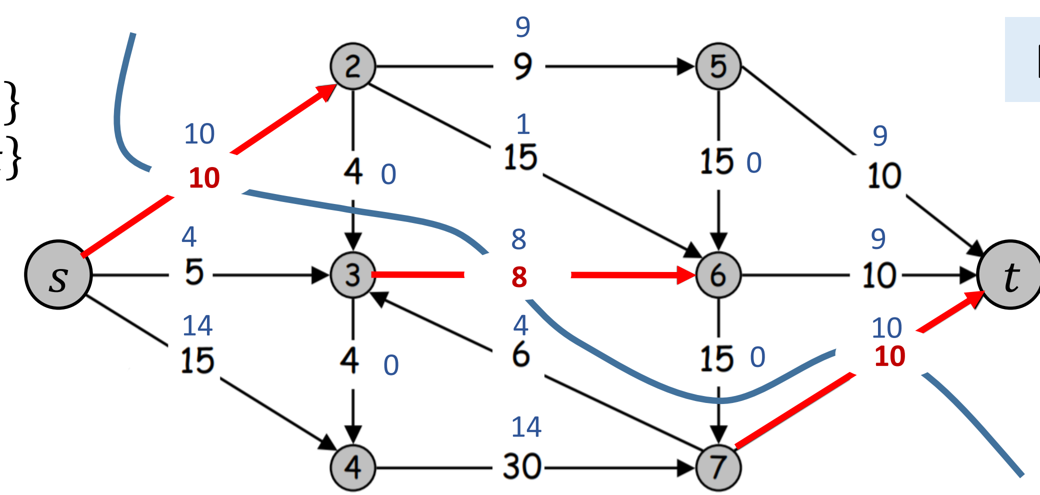
Flows and Cuts

- **Observation 3**: Let f be a flow, and let (S, T) be an s-t cut whose capacity equals the value of f. Then, f is a max flow and (S, T) is a min cut.

Example:

Cut Capacity = 28 $\Rightarrow$ Flow value $\leq$ 28

$S = \{s, v_3, v_4, v_7\}$
$T = \{v_2, v_5, v_6, t\}$



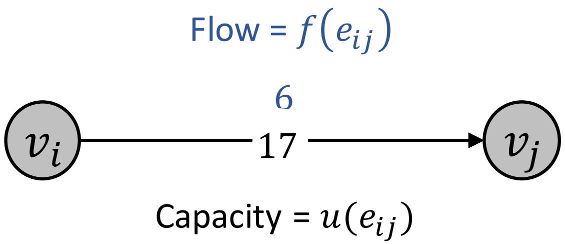
Flow value = 28

Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

Residual Graphs

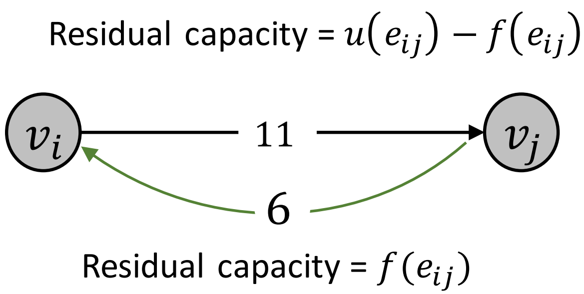
Original Graph:

- Flow $f(e_{ij})$
- Edge $e_{ij} =: v_i - v_j$



Residual edge:

- Edge $e_{ij} =: v_i - v_j$ or $e_{ji} =: v_j - v_i$
- "undo" flow sent



Residual graph:

- All the edges that have strictly positive residual capacity.

Slide adapted from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

Augmenting Paths

Observation 4: If any augmenting path exits, then not yet a max flow.

Augmenting path = path in residual graph

Examples:

Augmenting paths exist, **not yet** a max flow! | No augmenting path, a **max flow**!

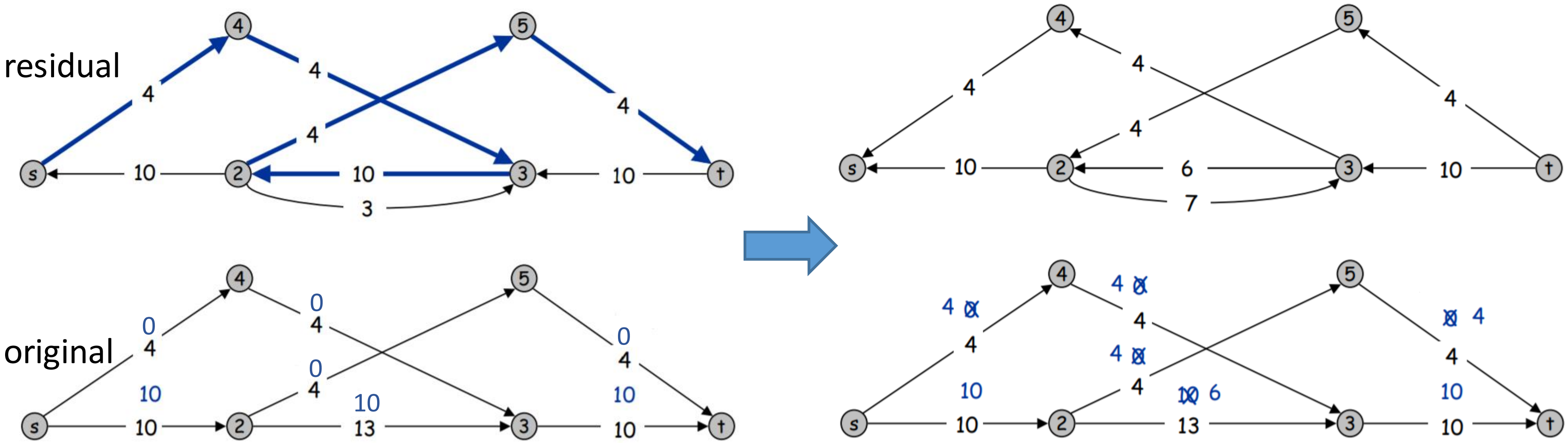


Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

Augmenting Path Algorithm

Augmenting Path algorithm:

while (there exists an augmenting path) {

1. Find augmenting path P
2. Compute bottleneck capacity of P // lowest edge capacity in P
3. Augment flow along P

}

Choosing Good Augmenting Paths:

1. Fewest number of arcs (shortest path), easy to implement with Breadth-First-Search.
2. Max bottleneck capacity (fattest path), use Dijkstra-style (Best-First-Search) algorithm.

Max-Flow Min-Cut Theorem

- **Max-Flow Min-Cut Theorem** (Ford-Fulkerson, 1956): In any network, the value of max-flow equals capacity of min-cut.

$\Rightarrow$ we find flow and cut such that Observation 3 applies.

Example:

Cut Capacity = 28 $\Leftrightarrow$ Flow value = 28

$S = \{s, v_3, v_4, v_7\}$
$T = \{v_2, v_5, v_6, t\}$

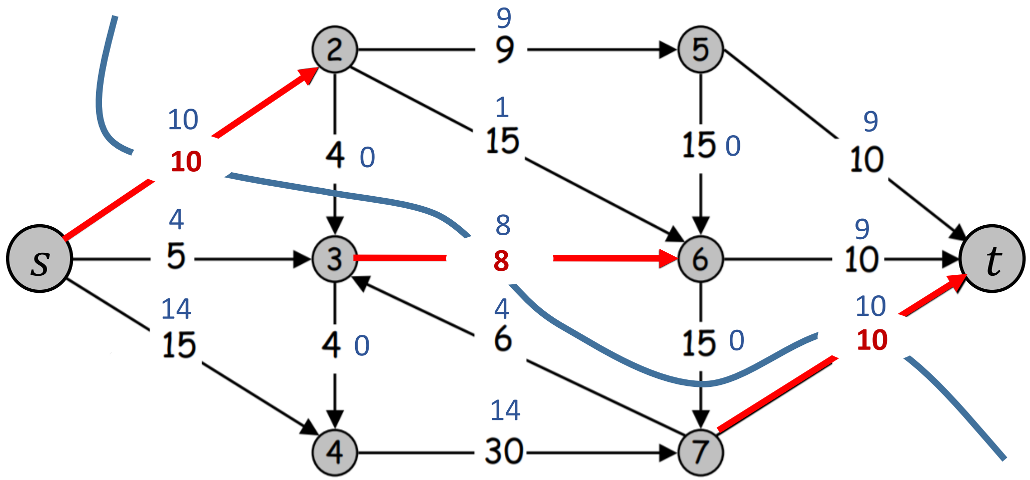


Image modified from: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

Max-Flow Min-Cut Theorem

- **Augmenting Path Theorem**: A flow f is a max flow if and only if there are no augmenting paths.

- **Max-Flow Min-Cut Theorem**: The value of the max-flow is equal to the capacity of the min-cut.

Max-Flow Min-Cut Theorem: Proof

We prove the augmenting paths and max-flow min-cut theorems simultaneously by showing the following are equivalent:

i. f is a max-flow

ii. There is no augmenting path relative to f

iii. There exists a cut whose capacity equals the value of f

(i) ⇒ (ii) : equivalent to not (ii) ⇒ not (i), which is Observation 4

(iii) ⇒ (i) : which is Observation 3

(ii) ⇒ (iii) : if no augmenting path relative to f, then there exists a cut whose capacity equals the value of f

Max-Flow Min-Cut Theorem: Proof

Proof: (ii)⇒(iii) if no augmenting path relative to f, then there exists a cut whose capacity equals the value of f

Let f be a flow with no augmenting paths, and S be set of vertices reachable from s in residual graph:

- $s \in S$ and no augmenting paths ⇒ $t \notin S$
- all edges e **leaving** S in original network have $f(e) = u(e)$
- all edges e **entering** S in original network have $f(e) = 0$

$$|f| = \sum_{e \text{ } out \text{ } of \text{ } S} f(e) - \cancelto{0}{\sum_{e \text{ } in \text{ } to \text{ } S} f(e)}$$
$$= \sum_{e \text{ } out \text{ } of \text{ } S} u(e)$$
$$= \text{capacity}(S, T) \qquad \square$$

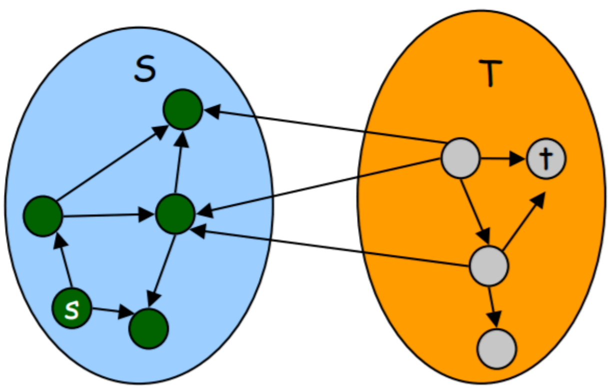


Image source: http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

Augmented Path Algorithm: Example

Two numbers represent: current flow / edge capacity

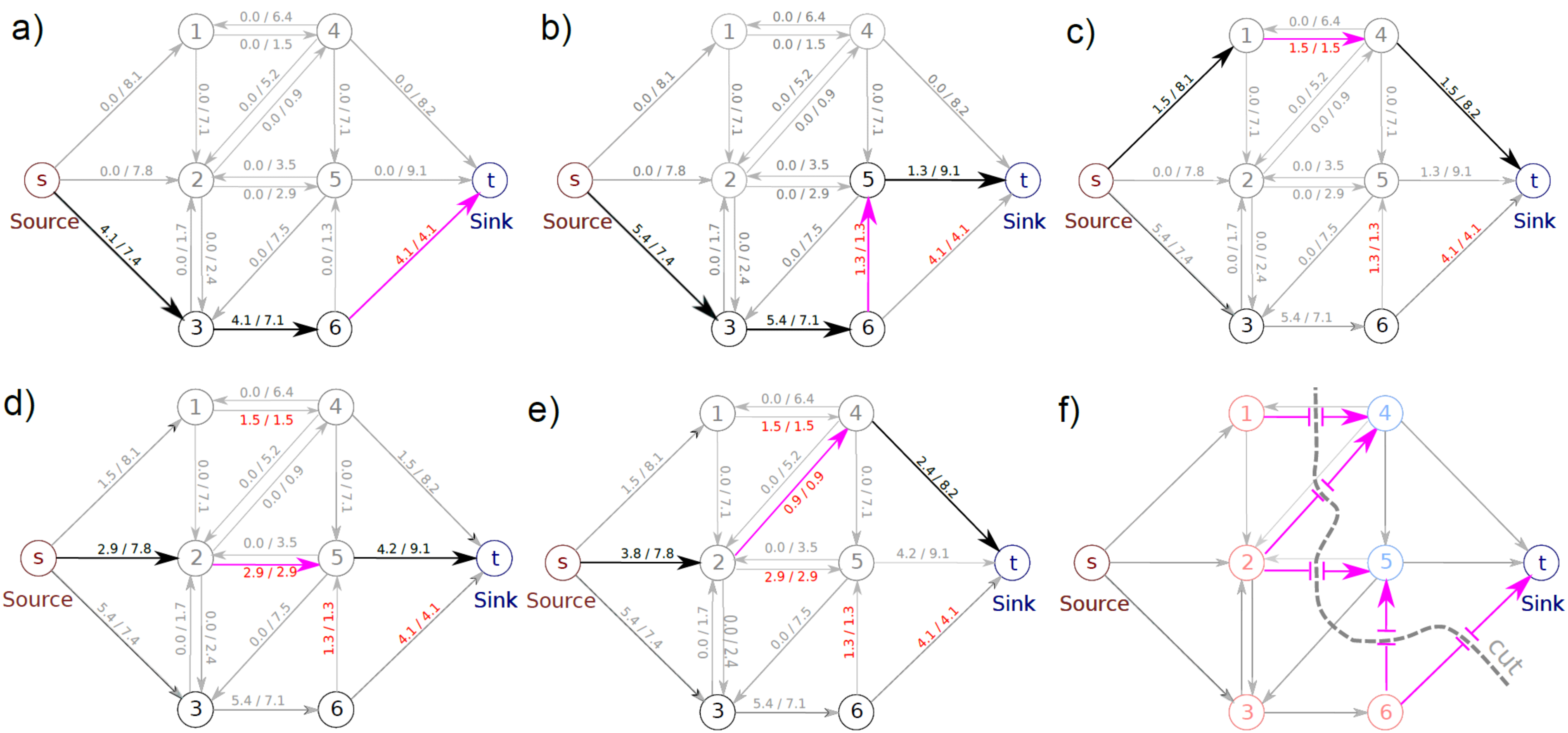


Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Case 1: Binary MRF

- Graph-cuts used to optimize this cost function:

$$\underset{w_{1\ldots N}}{\arg\min} \boxed{\sum_{n=1}^{N} U_n(w_n)} + \boxed{\sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n)},$$

Unary terms
(compatibility of data with label w)

Pairwise terms
(compatibility of neighboring labels)

- Binary case: $w_n \in \{0,1\}$

- Constrain pairwise costs so that they are "zero-diagonal"

$$P_{m,n}(0,0) = 0 \qquad P_{m,n}(1,0) = \theta_{10}$$

$$P_{m,n}(0,1) = \theta_{01} \qquad P_{m,n}(1,1) = 0,$$

Binary MRF: Graph Construction

- One vertex per pixel and neighbors in the pixel grid are connected by reciprocal pairs of directed edges.
- Each pixel vertex receives a connection from the source and sends a connection to the sink.
- To separate source from sink, the cut must include one of these two edges for each vertex.
- The choice of which edge is cut will determine which of two labels is assigned to the pixel.

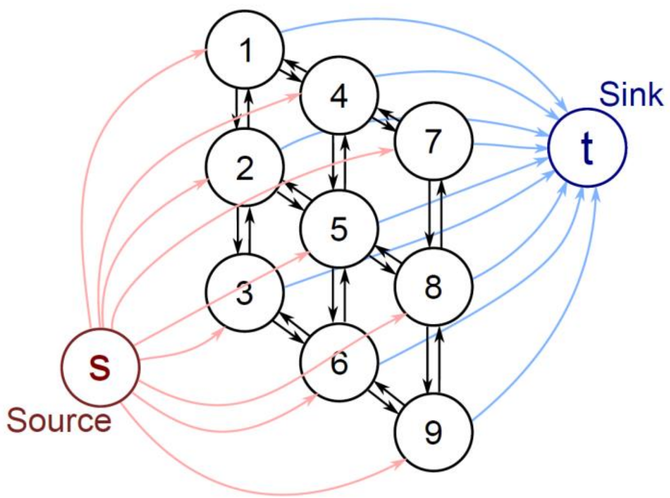


Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Binary MRF: Graph Construction

- Add capacities so that minimum cut, minimizes the cost function.

- Unary costs $U(0), U(1)$ attached to links to source and sink; either one or the other is paid.

- Pairwise costs $P_{ij}(0{,}1), P_{ij}(1{,}0)$ between pixel nodes; either one has to be included when pixels i and j takes opposite labels.

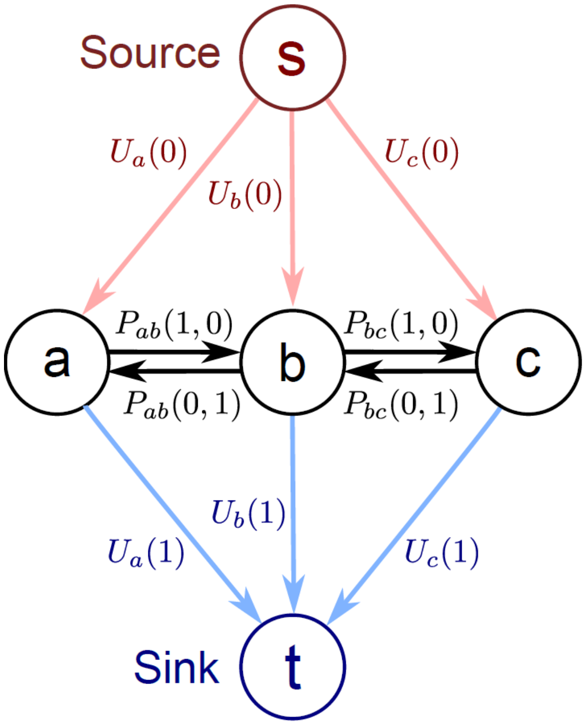
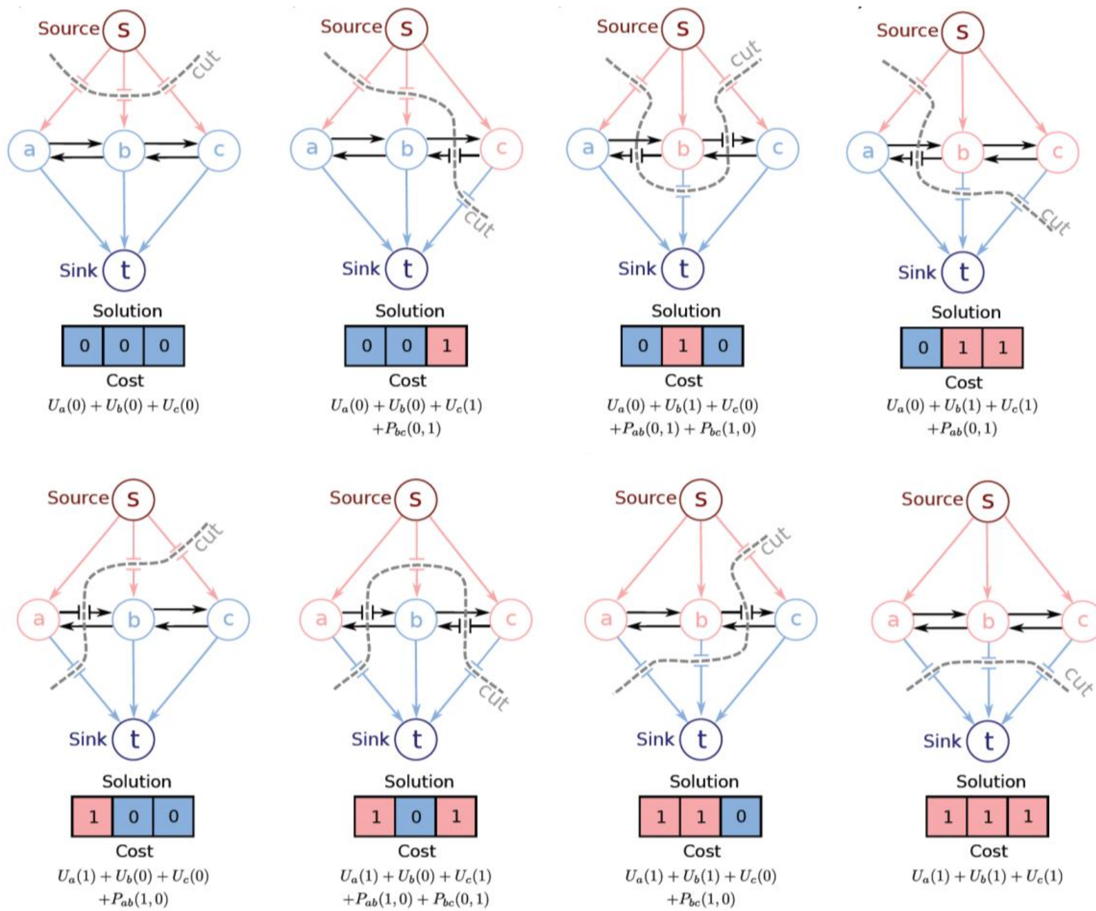


Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

MAP Inference

Augmenting Path algorithm will find the Min-Cut solution, which is equivalent to the minimizing the cost (i.e. MAP solution).



$$\underset{w_{1\ldots N}}{\operatorname{argmin}} \sum_{n=1}^{N} U_n(w_n) + \sum_{(m,n)\in\mathcal{C}} P_{mn}(w_m, w_n),$$

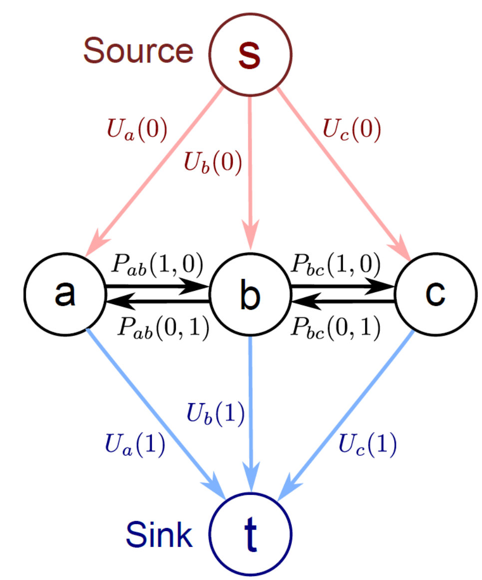


Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

General Pairwise Costs

$$P_{m,n}(0,0) = \boxed{\theta_{00}} \qquad P_{m,n}(1,0) = \theta_{10}$$
$$P_{m,n}(0,1) = \theta_{01} \qquad P_{m,n}(1,1) = \boxed{\theta_{11}}$$

No longer zero cost!

- Modify graph to:

1. Add $P(0,0)$ to edge s-b; implies that solutions 0,0 and 1,0 also pay this cost.
2. Subtract $P(0,0)$ from edge a-b; solution 1,0 has this cost removed.

- Similar approach for $P(1,1)$.

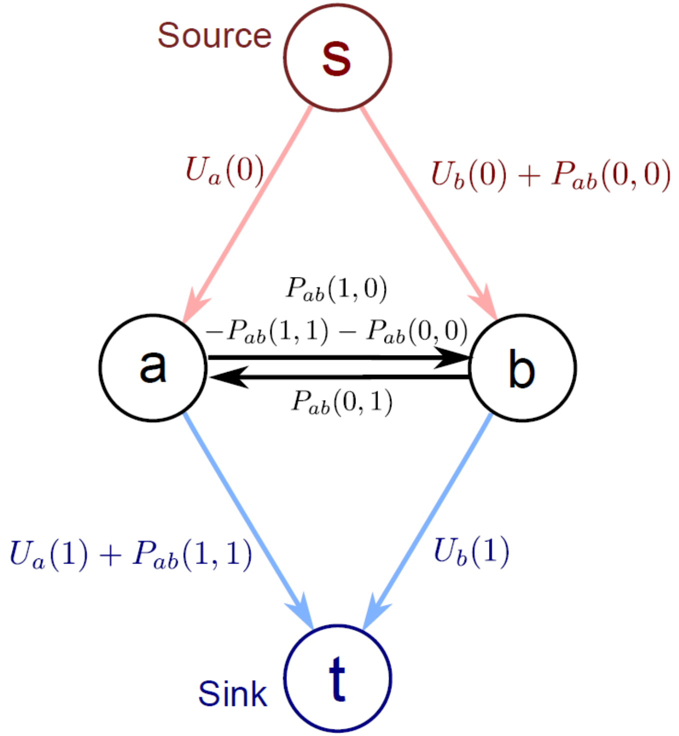


Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Re-parameterization

Problem:

- The max-flow / min-cut algorithm requires the capacities of all edges to be non-negative.

- However, we cannot guarantee $P_{ab}(1,0) - P_{ab}(1,1) - P_{ab}(0,0)$ to be non-negative!

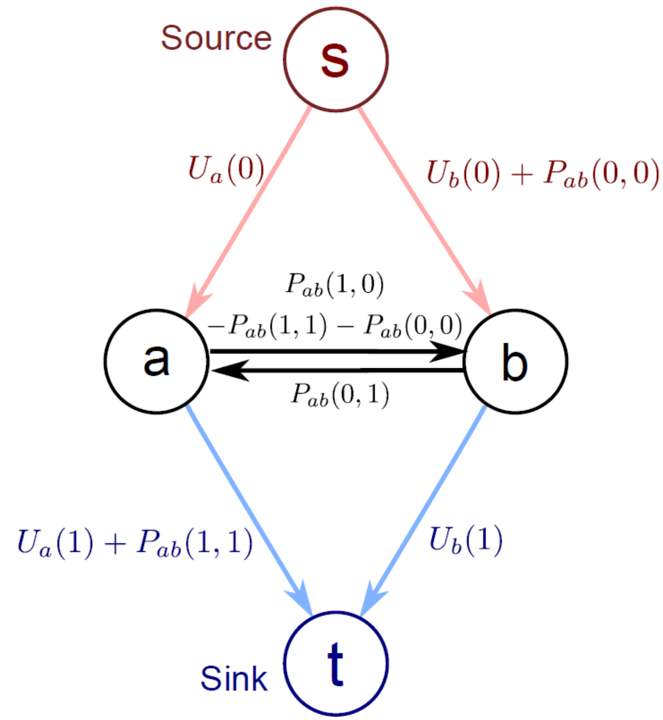


Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Re-parameterization

Solution:

- Adjust the edge capacities so that every possible solution has a constant cost β added to it.
- MAP solution remains unchanged.

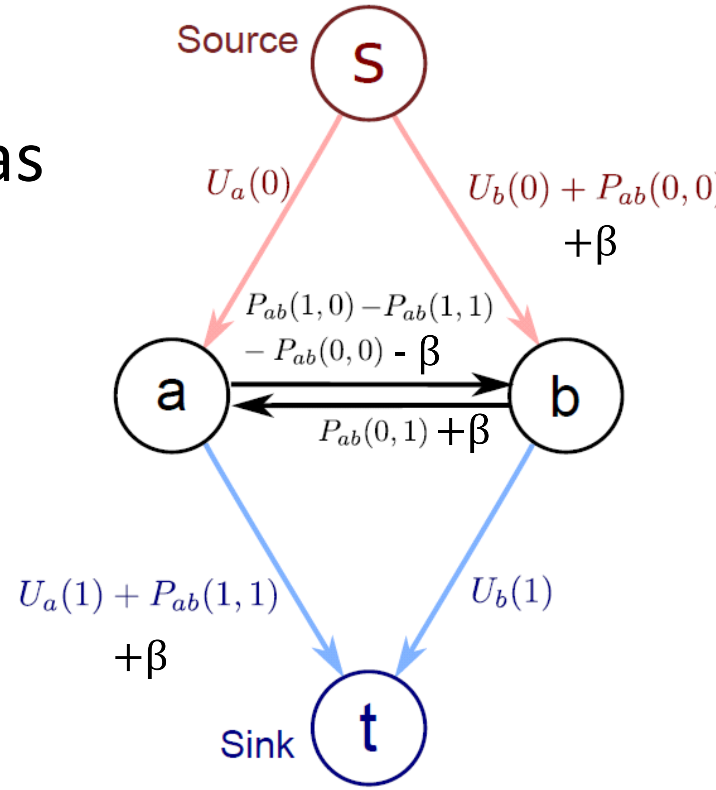


Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Submodularity

$$P_{m,n}(0,0) = \theta_{00} \qquad P_{m,n}(1,0) = \theta_{10}$$
$$P_{m,n}(0,1) = \theta_{01} \qquad P_{m,n}(1,1) = \theta_{11}$$

For edge capacities between a and b to be positive:

$$\begin{aligned} \theta_{10} - \theta_{11} - \theta_{00} - \beta &\geq 0 \\ \theta_{01} + \beta &\geq 0 \end{aligned}$$

Adding together implies:

$$\boxed{\theta_{10} + \theta_{01} - \theta_{11} - \theta_{00} \geq 0}$$

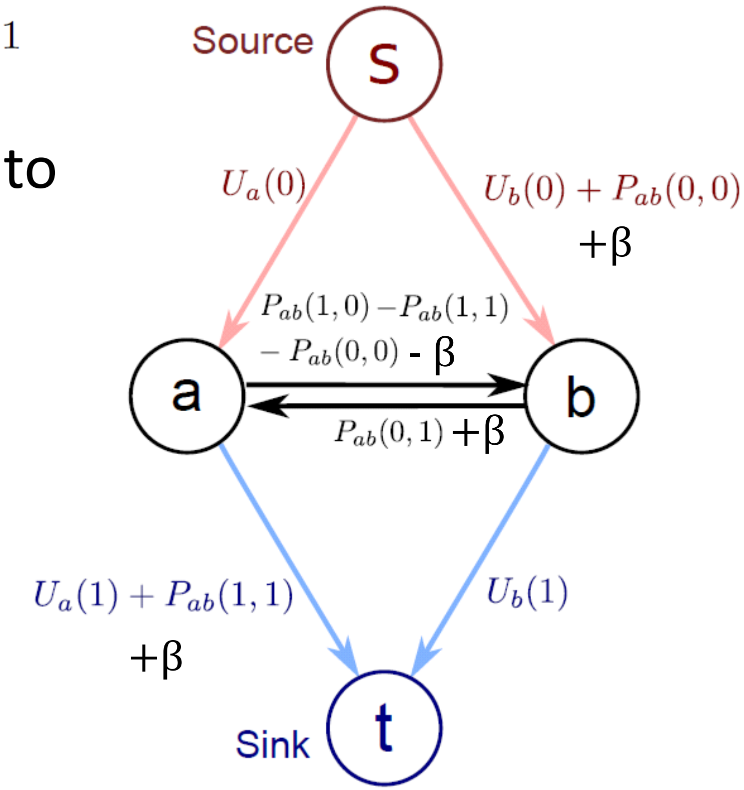


Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Submodularity

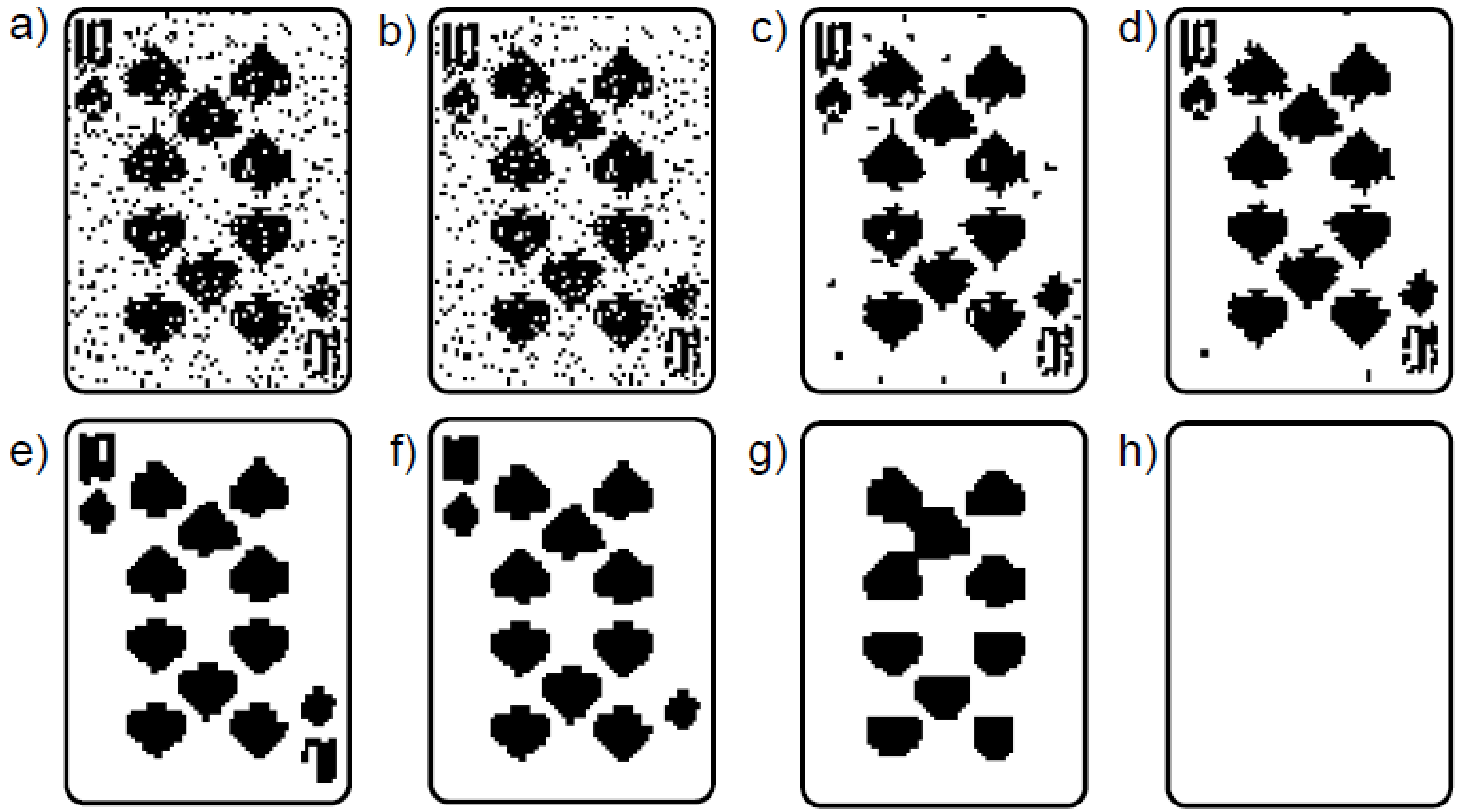
$$\theta_{10} + \theta_{01} - \theta_{11} - \theta_{00} > 0$$

- If this condition is obeyed, it is said that the problem is "submodular" and it can be solved in polynomial time.
- Otherwise, the problem is NP-hard.
- Usually not a problem as we tend to favour smooth solutions, i.e. θ_{11} and $\theta_{00} \ll \theta_{10}$ and θ_{01}.

Denoising Results

Original

Pairwise costs increasing

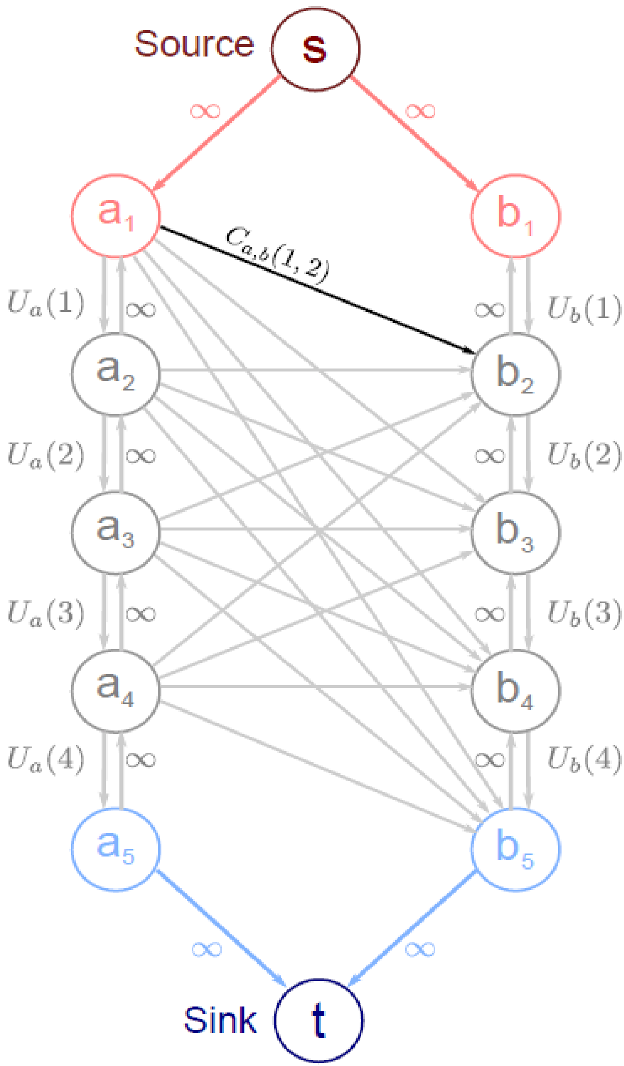


Pairwise costs increasing

Over-smoothing

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

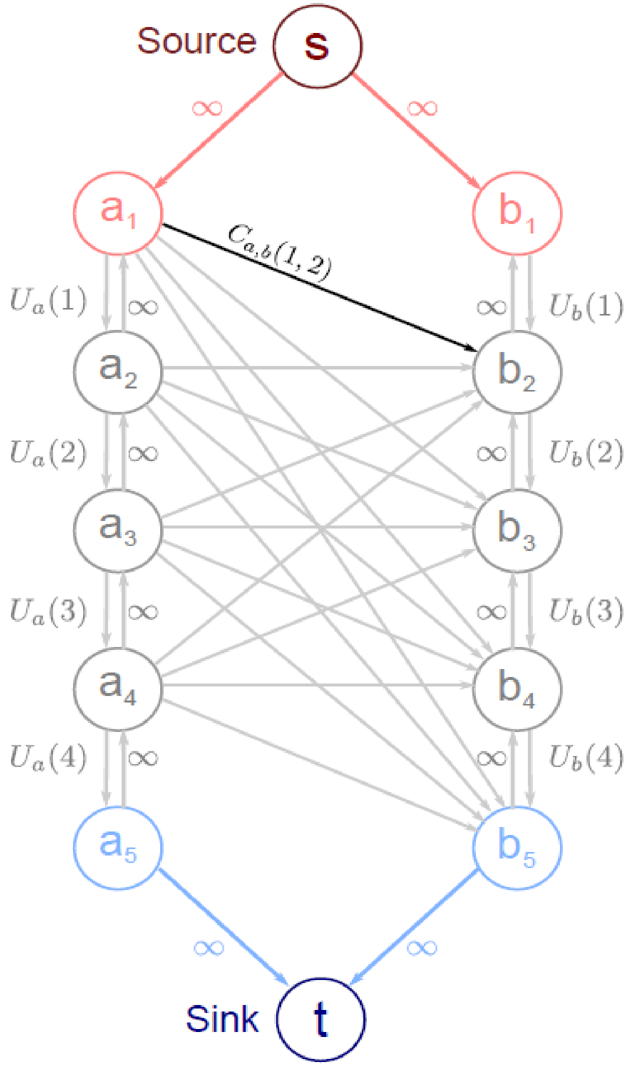
Case 2: Multiple Labels (Submodular)



- Multi-label case: $w_n \in \{0,1,\dots,K\}$.
- With K labels and N pixels, we add $(K+1)N$ vertices into the graph.
- For each pixel, the $K+1$ associated vertices are stacked.
- The top and bottom of the stack are connected to the source and sink by edges with infinite capacity.

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

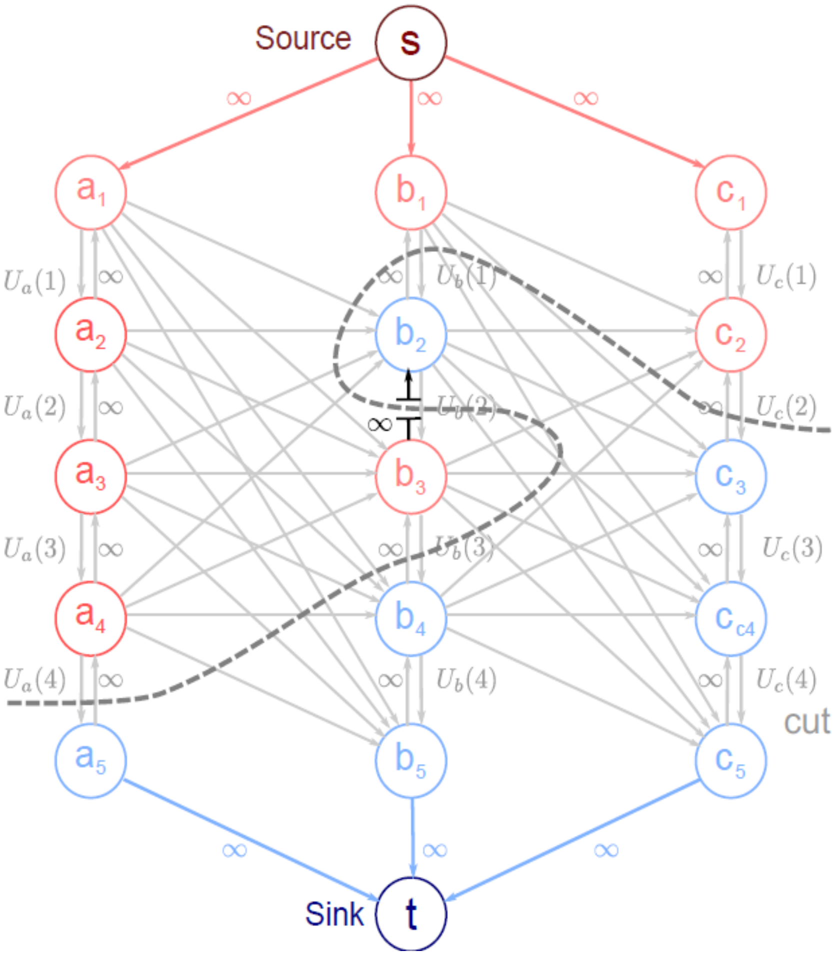
Case 2: Multiple Labels (Submodular)



- Between the $K + 1$ vertices in the stack are K edges forming a path from source to sink.
- These edges are associated with the K unary costs $U_n(1), \ldots, U_n(K)$.
- Cut at one of the K edges in this chain to separate source from sink.
- A cut at the k^{th} edge in this chain indicates the pixel takes label k and this incurs the cost of $U_n(k)$.

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

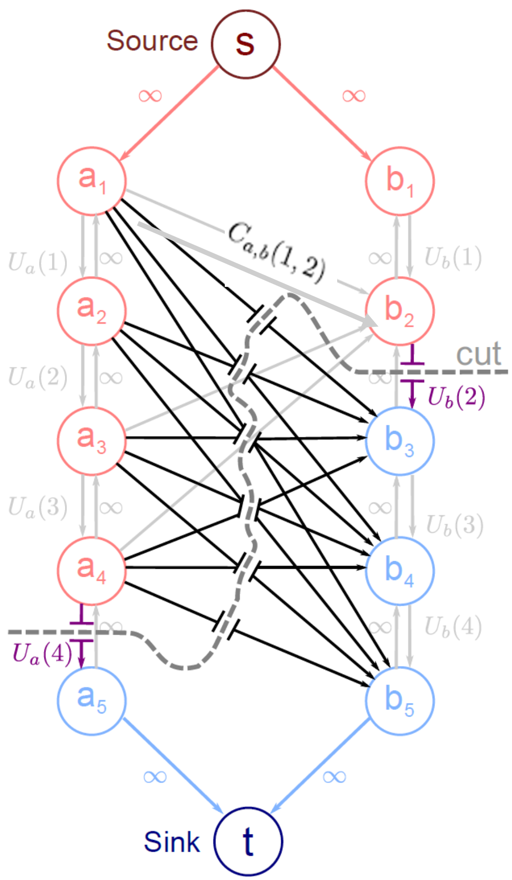
Constraint Edges



- Add constraint edges to ensure only a single edge from the chain is part of the minimum cut (i.e., each cut is one valid labeling).
- The constraint edges connect the vertices backwards along each chain.
- Any cut that crosses the chain more than once must cut one of these edges and will never be the minimum cut solution.

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Pairwise Cost



If pixel a takes label I and pixel b takes label J, the cost of this cut is given by:

$$U_a(I) + U_b(J) + \sum_{i=1}^{I} \sum_{j=J+1}^{K+1} C_{ab}(i,j),$$

where

$$\begin{aligned}\sum_{i=1}^{I} \sum_{j=J+1}^{K+1} C_{ab}(i,j) &= \sum_{i=1}^{I} \sum_{j=J+1}^{K+1} P_{ab}(i,j-1)+P_{ab}(i-1,j)-P_{ab}(i,j)-P_{ab}(i-1,j-1)\\ &= P_{ab}(I,J)+P_{ab}(0,J)-P_{ab}(I,K+1)-P_{ab}(0,K+1)\\ &= P_{ab}(I,J).\end{aligned}$$

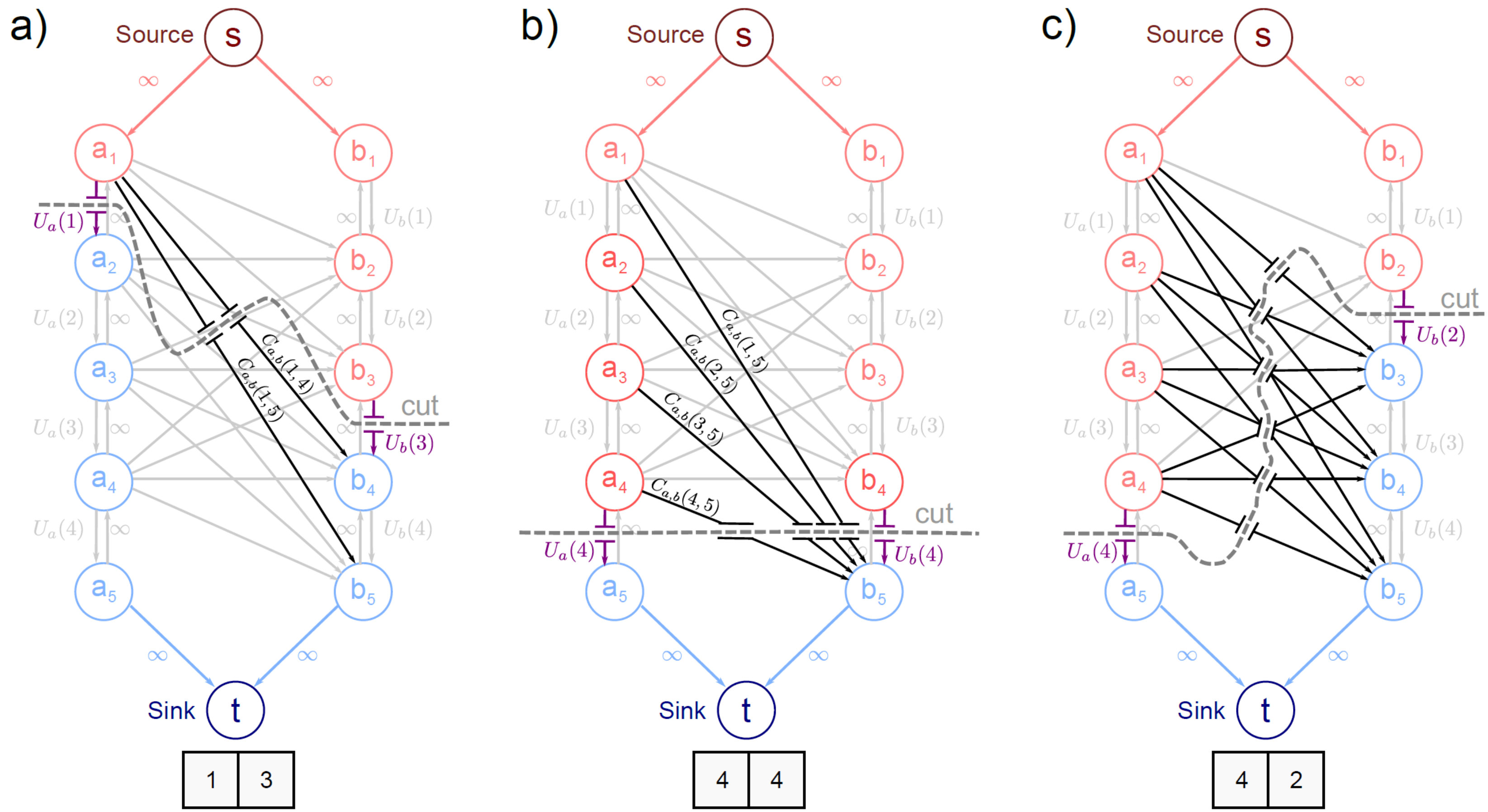
Superfluous pairwise costs associated with the non-existent labels 0 or K + 1 defined to be zero:

$$\begin{array}{lll} P_{ab}(i,0)=0 & P_{ab}(i,K+1)=0 & \forall\ i \in \{0\ldots K+1\}\\ P_{ab}(0,j)=0 & P_{ab}(K+1,j)=0 & \forall\ j \in \{0\ldots K+1\}. \end{array}$$

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Case 2: Multiple Labels (Submodular)

Example Cuts:



Cost
$U_a(1) + U_b(3) + \sum_{i=1}^{1}\sum_{j=4}^{5} C_{ab}(i,j)$

Cost
$U_a(4) + U_b(4) + \sum_{i=1}^{4}\sum_{j=5}^{5} C_{ab}(i,j)$

Cost
$U_a(4) + U_b(2) + \sum_{i=1}^{4}\sum_{j=3}^{5} C_{ab}(i,j)$

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Submodularity

- We require the remaining inter-pixel links to be positive so that:

$$C_{ab}(i,j) \geq 0$$

or

$$P_{ab}(i,j-1) + P_{ab}(i-1,j) - P_{a,b}(i,j) - P_{ab}(i-1,j-1) \geq 0$$

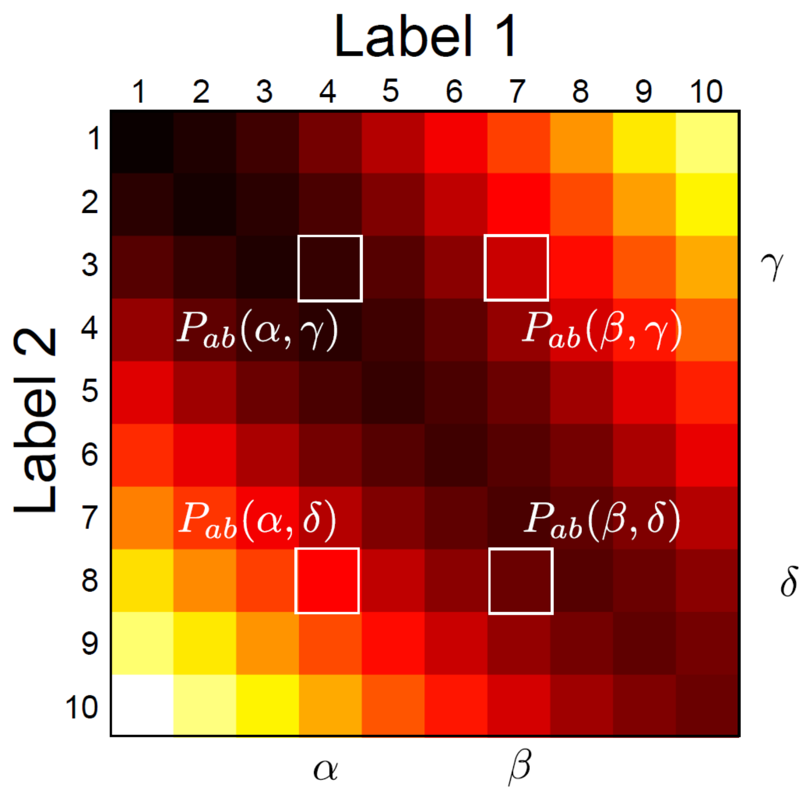
- By mathematical induction we can get the more general result:

$$P_{ab}(\beta,\gamma) + P_{ab}(\alpha,\delta) - P_{a,b}(\beta,\delta) - P_{ab}(\alpha,\gamma) \geq 0,$$

- (α, β) and (γ, δ) are any two-label pairs.

Submodularity

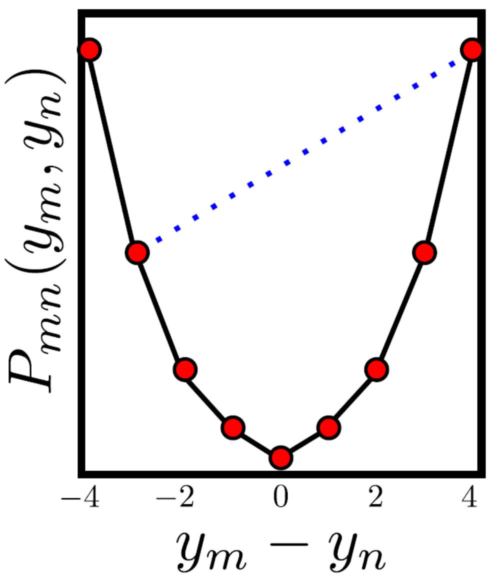
$$P_{ab}(\beta,\gamma) + P_{ab}(\alpha,\delta) - P_{a,b}(\beta,\delta) - P_{ab}(\alpha,\gamma) \geq 0,$$



If not submodular, then the problem is NP-hard!

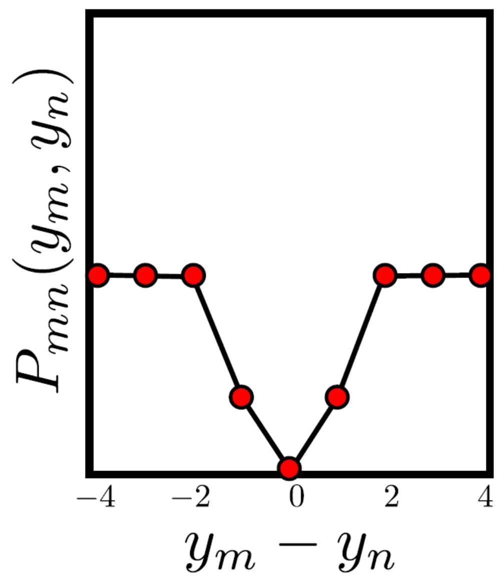
Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Convex vs. Non-Convex Costs



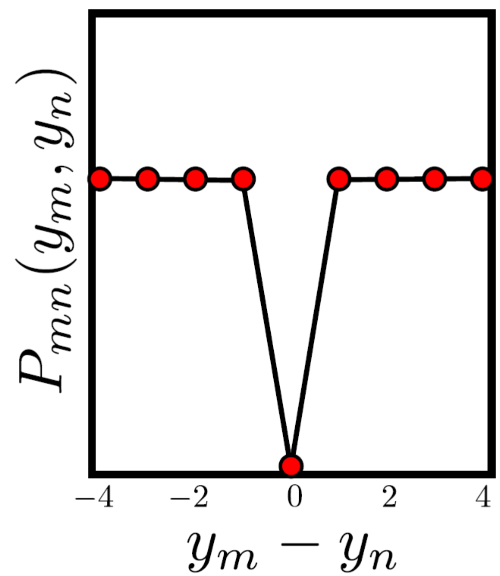
Quadratic

- Convex
- Submodular



Truncated Quadratic

- Not Convex
- Not Submodular

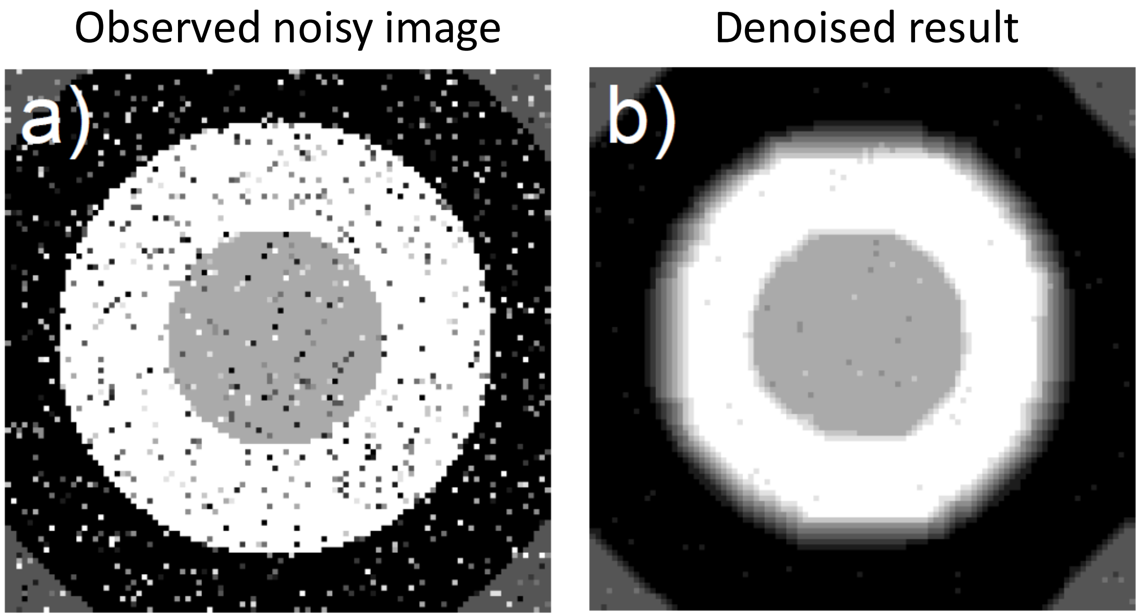


Potts Model

- Not Convex
- Not Submodular

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

What's Wrong with Convex Costs?



- Pay lower price for many small changes than one large one
- Result: blurring at large changes in intensity

Non-convex costs are preferred!

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

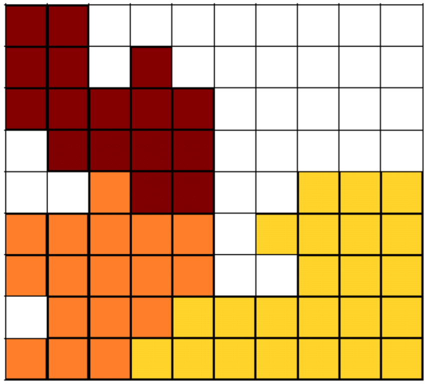
Case 3: Multiple Labels (Non-Submodular)

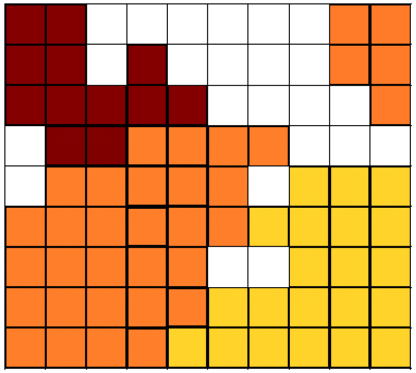
Alpha-Expansion Algorithm:

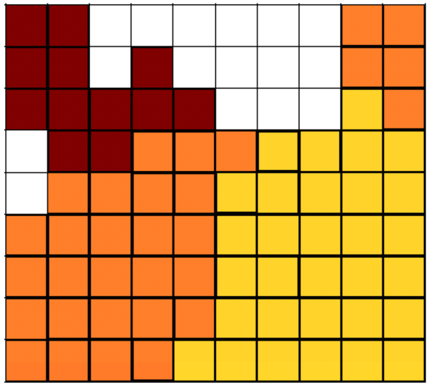
- Breaks the problem down into a series of binary sub-problems.
- At each step, we choose a label α and expand: for each pixel (with other labels) we either leave the label as it is, or replace it with α.
- Terminating condition: The process is iterated until no choice of α causes any change.
- Each expansion move is guaranteed to lower the overall cost, although global optimality not guaranteed.

Case 3: Multiple Labels (Non-Submodular)

Example:

a)

Initial labeling

b)

Orange label is expanded: each label stays the same or becomes orange.

c)

Yellow label is expanded

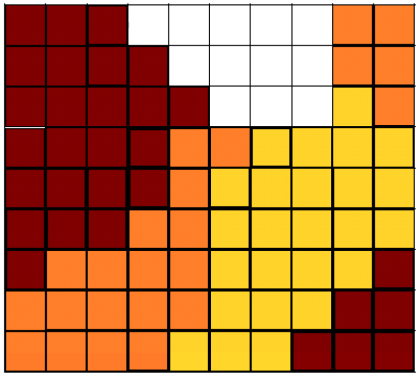
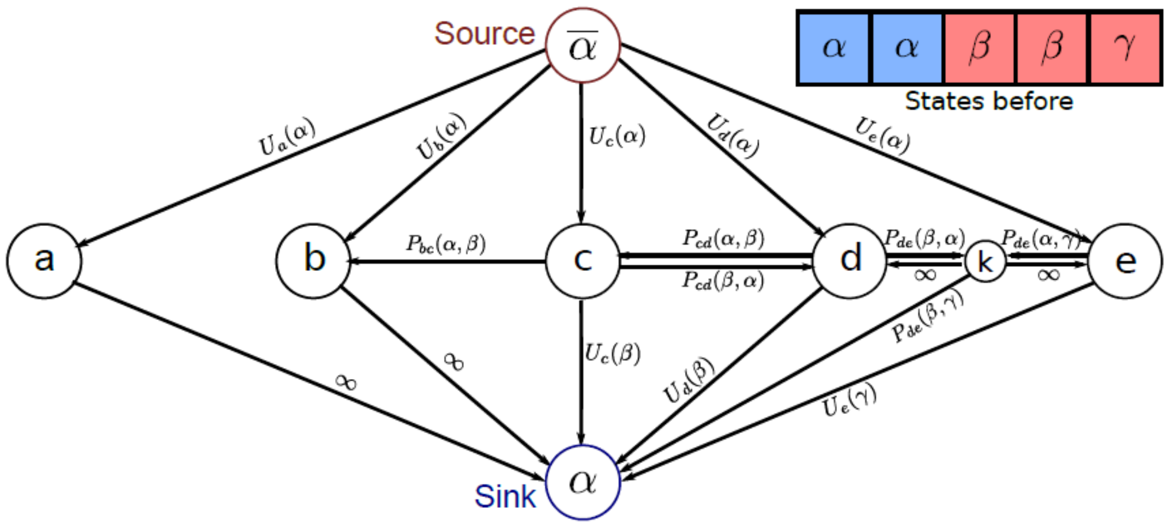
d)

Red label is expanded

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

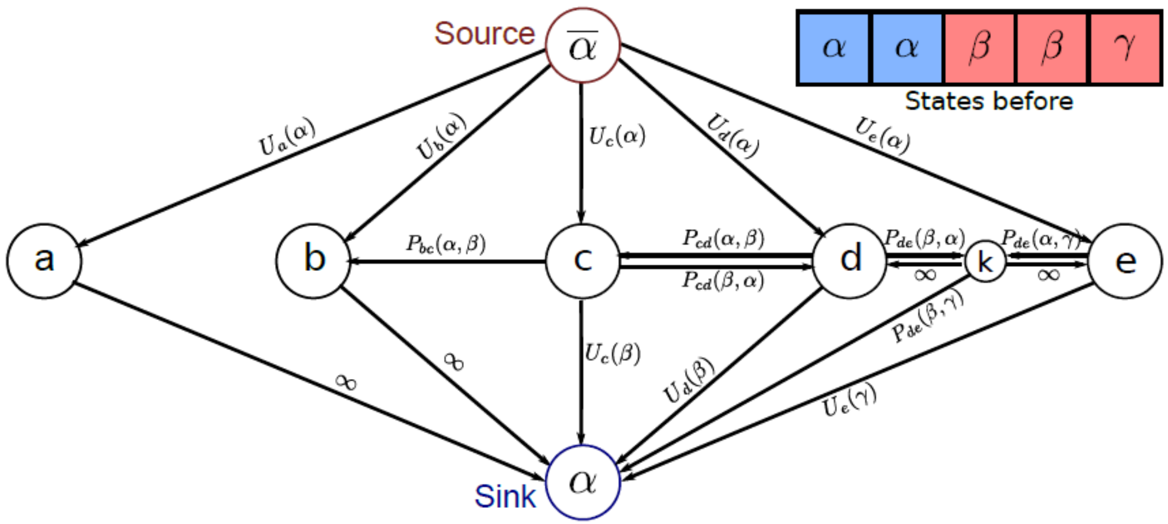
Alpha-Expansion: Graph Construction



- One vertex associated with each pixel.
- Each vertex is connected to the source and sink (i.e. keep the original label $\bar{\alpha}$ or change to α).
- To separate source from sink, we must cut one of these two edges at each pixel.

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Alpha-Expansion: Graph Construction

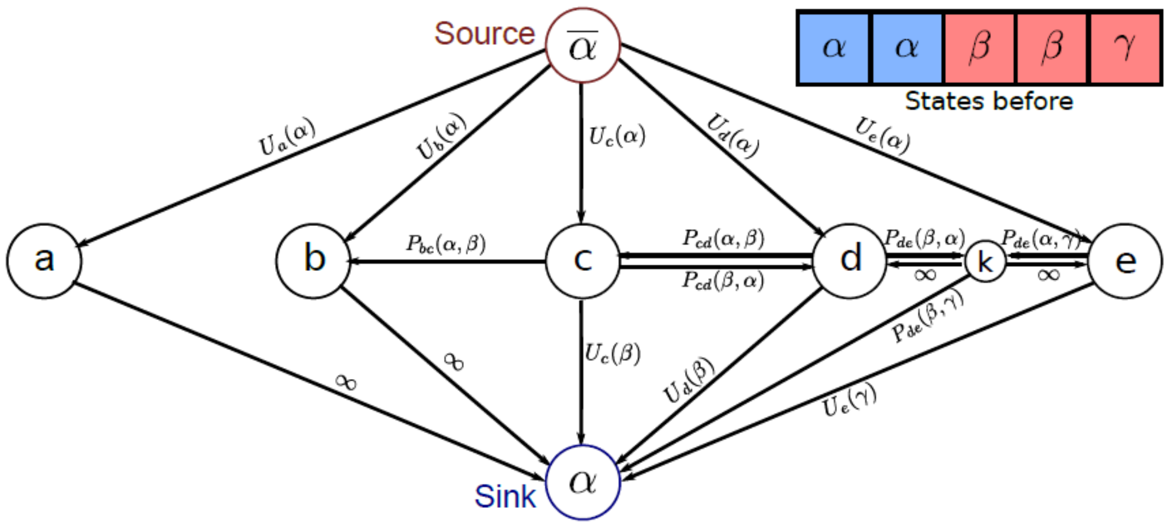


- We associate the unary costs for each edge being set to or its original label with the two links from each pixel.

- If the pixel already has label α, then we set the cost of being set to $\bar{\alpha}$ to ∞.

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

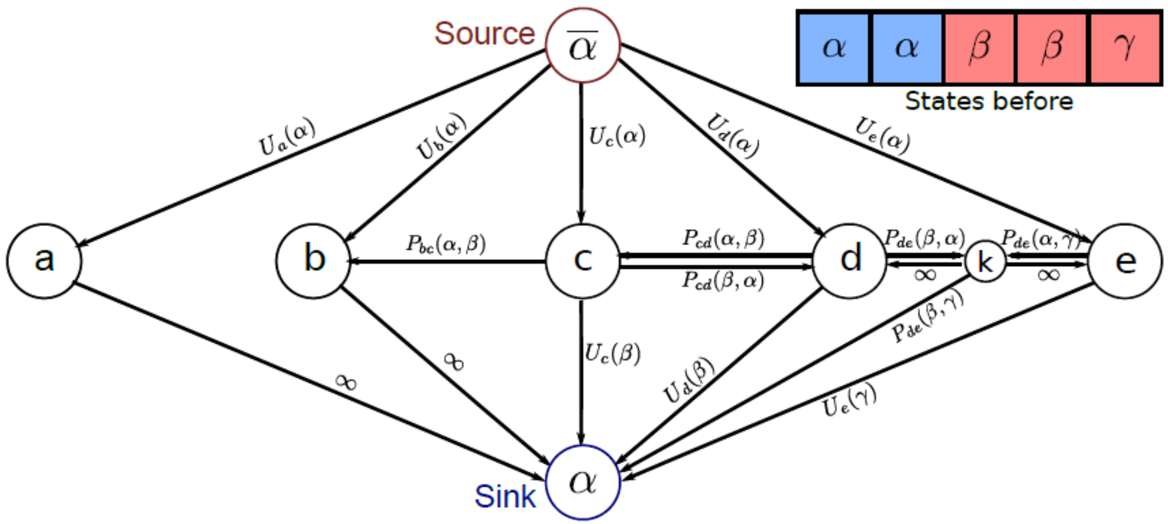
Alpha-Expansion: Graph Construction



- We associate the unary costs for each edge being set to or its original label with the two links from each pixel.
- If the pixel already has label α, then we set the cost of being set to $\bar{\alpha}$ to ∞.
- Remaining structure of graph is dynamic: it changes at each iteration depending on the choice of α and the current labels.

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

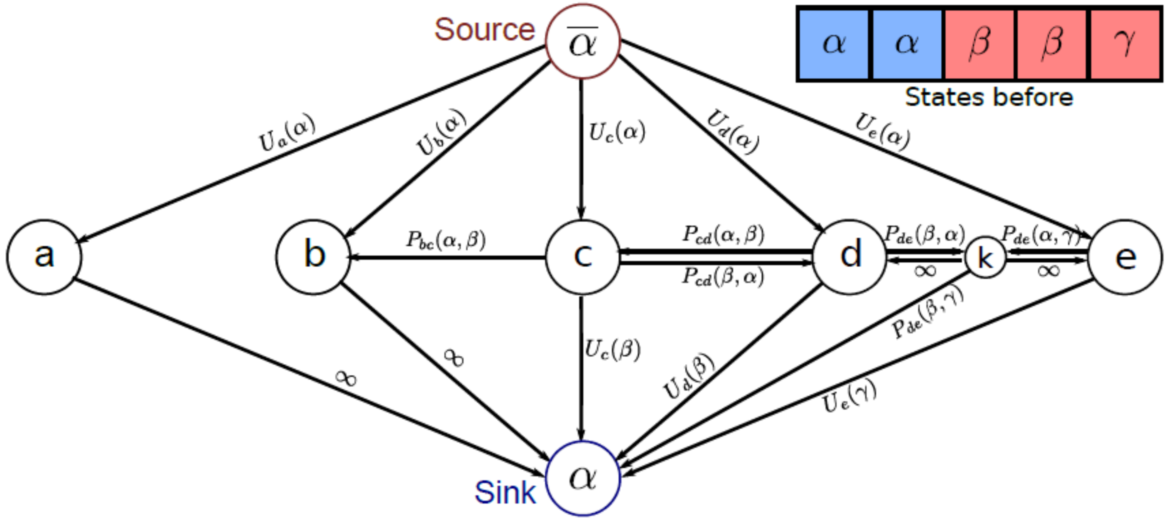
Alpha-Expansion: Graph Construction



There are four possible relationships between adjacent pixels, i.e. pairwise costs.

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Alpha-Expansion: Graph Construction

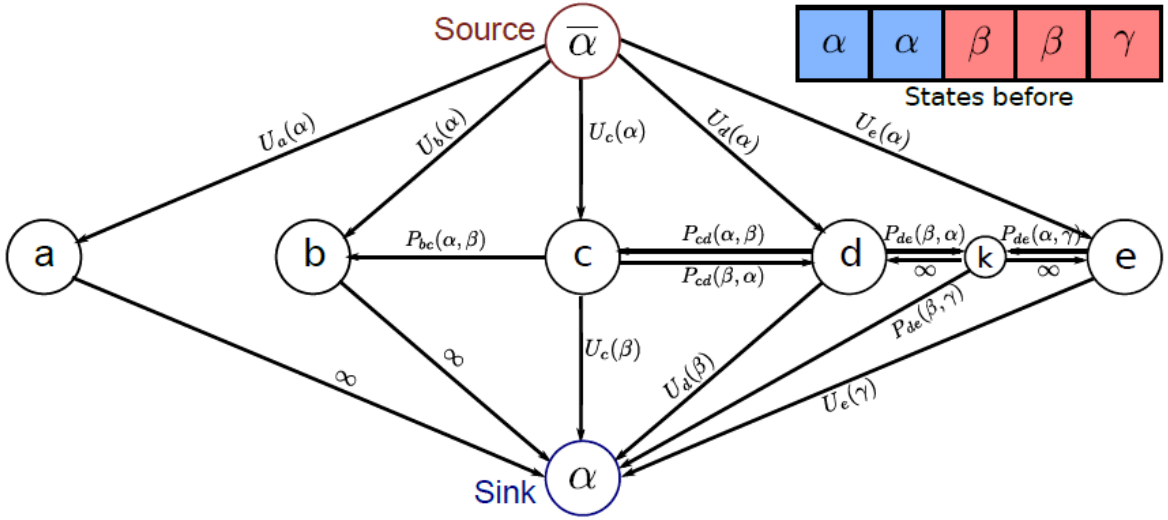


Relationship 1: Adjacent pixels i and j have label α.

- Final solution is inevitably $\alpha - \alpha$, and so the pairwise cost is zero – no extra edge
- See nodes a and b in the above example.

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Alpha-Expansion: Graph Construction

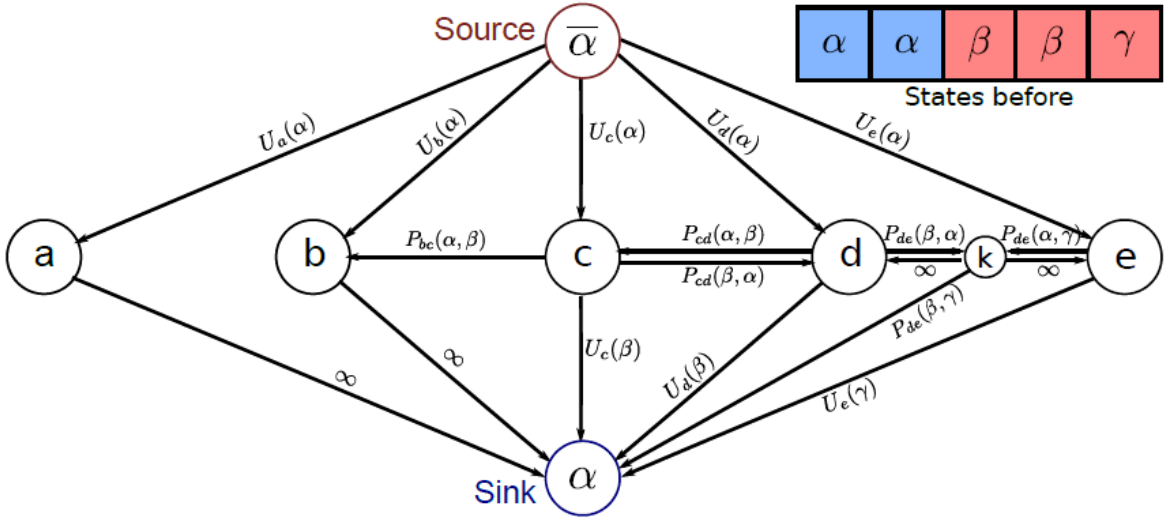


Relationship 2: Adjacent pixels i and j have labels α and β.

- Final solution may be:
  1. $\alpha - \alpha$ (no cost and no new edge)
  2. $\alpha - \beta$ ($P(\alpha, \beta)$, add new edge)
- See nodes b and c in the above example.

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Alpha-Expansion: Graph Construction

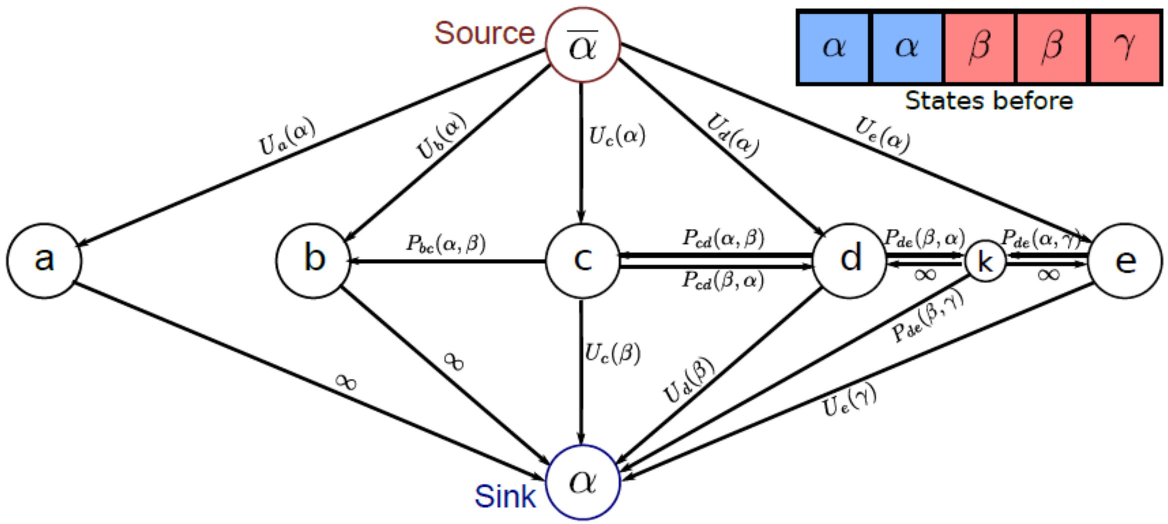


Relationship 3: Adjacent pixels i and j have the same label β.

- Final solution may:
  1. $\beta - \beta$ (no cost and no new edge)
  2. $\alpha - \beta$ ($\mathrm{P}(\alpha, \beta)$, add new edge)
  3. $\beta - \alpha$ ($\mathrm{P}(\beta, \alpha)$, add new edge)
- See nodes c and d in the above example.

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Alpha-Expansion: Graph Construction



Relationship 4: Adjacent pixels i and j have labels β and γ.

- Final solution may be:
  1. $\beta - \gamma$ ($\mathrm{P}(\beta, \gamma)$, add new edge between new vertex k and the sink)
  2. $\alpha - \gamma$ ($\mathrm{P}(\alpha, \gamma)$, add new edge between new vertex k and vertex j)
  3. $\beta - \alpha$ ($\mathrm{P}(\beta, \alpha)$, add new edge between vertex i and new vertex k)
  4. $\alpha - \alpha$ (no pairwise cost and no new edge)

- See nodes d and e in the above example.

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Triangle Inequality

- For the alpha-expansion algorithm to work, we require that the edge costs to form a metric, i.e.

$$P(\beta,\gamma) = 0 \Leftrightarrow \beta = \gamma$$
$$P(\beta,\gamma) = P(\gamma,\beta)$$
$$P(\beta,\gamma) \leq P(\beta,\alpha) + P(\alpha,\gamma)$$

Triangle Inequality

- Supposed that the triangle inequality does not hold, so that $P(\beta,\gamma) > P(\beta,\alpha) + P(\alpha,\gamma)$.
- We will get the wrong cut for $\beta - \gamma$, i.e. $d - k$ and $k - e$ will be cut instead of $k - \alpha$.

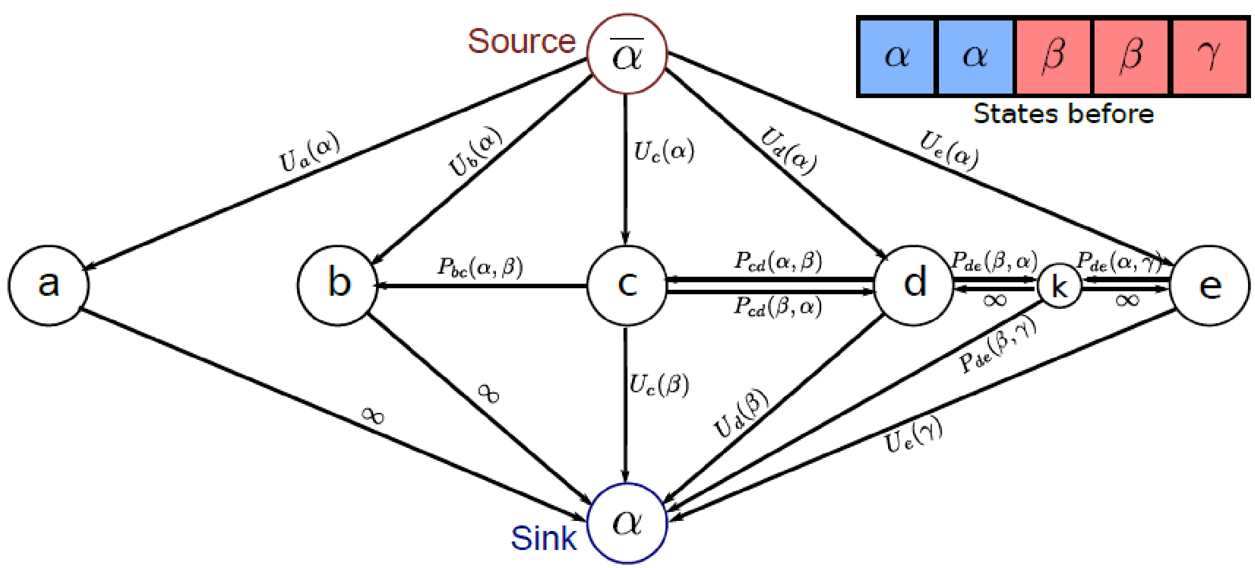
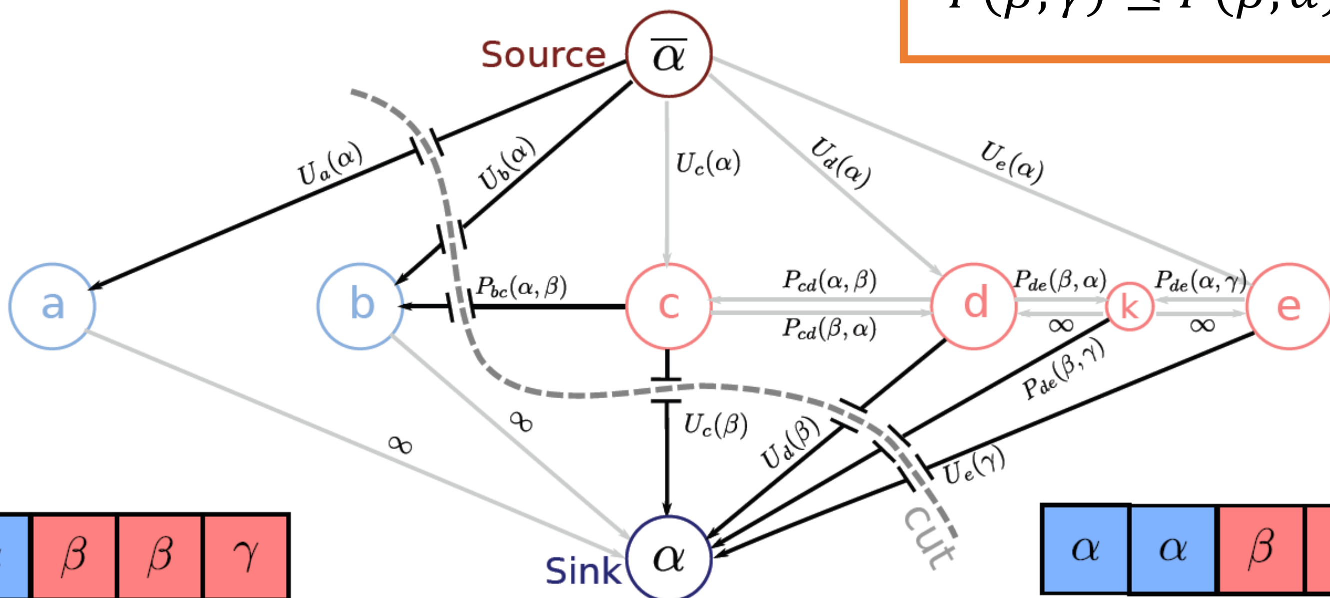


Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Example Cut 1

Important!

$$P(\beta,\gamma) \leq P(\beta,\alpha) + P(\alpha,\gamma)$$



α	α	β	β	γ

States before

α	α	β	β	γ

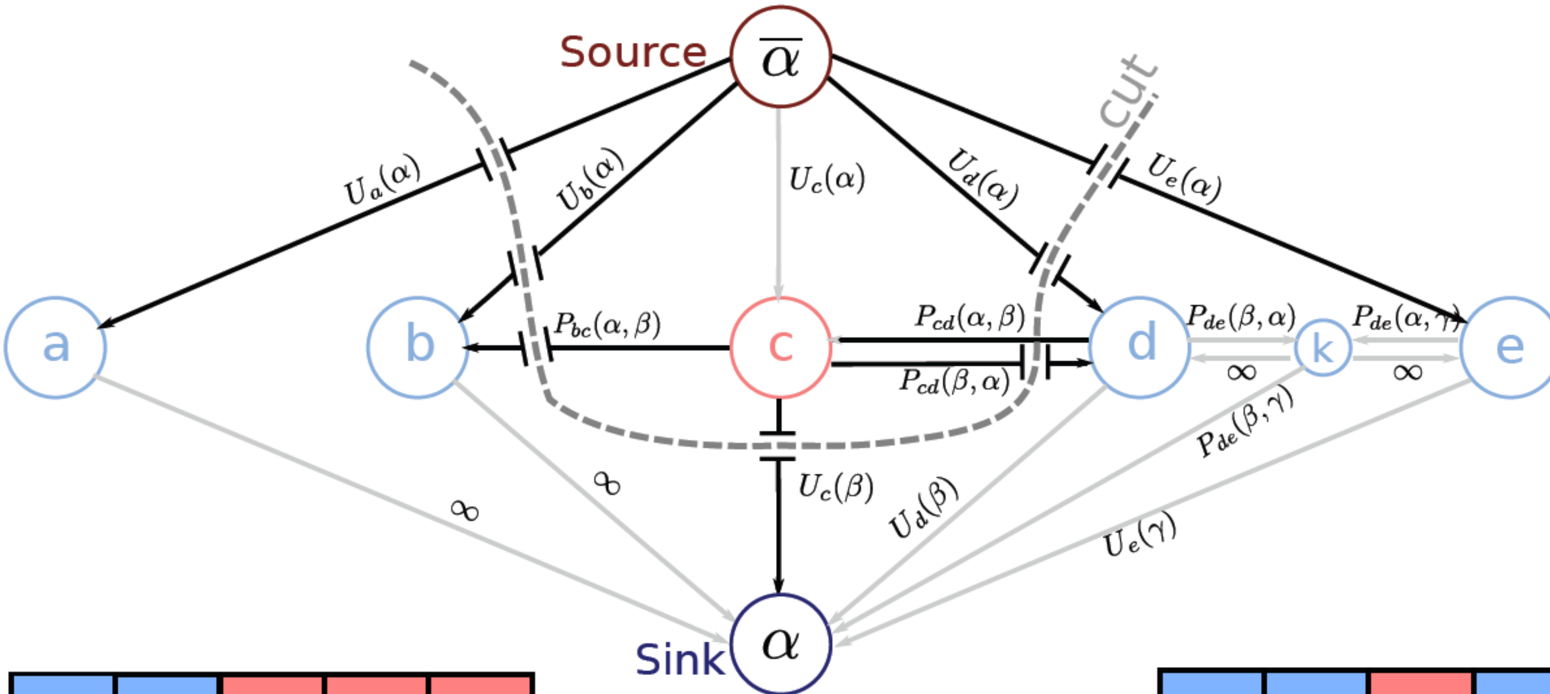
States after

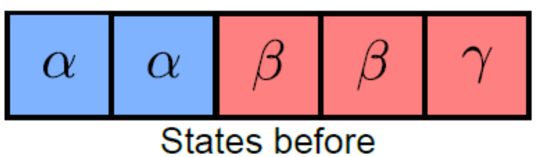
Cost

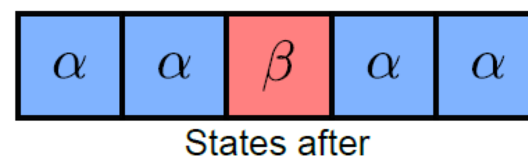
$$U_a(\alpha) + U_b(\alpha) + U_c(\beta) + U_d(\beta)$$
$$+U_e(\gamma) + P_{bc}(\alpha,\beta) + P_{de}(\beta,\gamma)$$

Image Source: "Computer vision: models, learning and inference", Simon J.D. Prince

Example Cut 2







Cost

$$U_a(\alpha)+U_b(\alpha)+U_c(\beta)+U_d(\alpha)$$
$$+U_e(\alpha)+P_{bc}(\alpha,\beta)+P_{cd}(\beta,\alpha)$$

Image Source: “Computer vision: models, learning and inference”, Simon J.D. Prince

Summary

- We have looked at:

1. Convert the binary/multi- labeling problem into a max-flow/min-cut problem.
2. Solve the max-flow/min-cut problem using the augmented path algorithm.
3. Explain the concept of sub-modularity.
4. Solve submodular binary/multi- labeling problem with max-flow/min-cut and non-submodular multi-labelling problem with alpha-expansion