CS6161 Midterm 2 Review

Jibang Wu

November 24, 2020

LP for mincost maxflow [Problem 5.1 in HW4]

1) use LP to compute maxflow value of the graph:
Let $x_{u,v}$ be the amount of flow in edge $u \to v$.
Maximize:

$$\sum_{(u,v)\in E} x_{u,v}$$

Subject to:

$$\forall v \notin \{s,t\}, \quad \sum_{u:(u,v)\in E} x_{u,v} - \sum_{u:(v,u)\in E} x_{v,u} = 0 \text{ (flow in } = \text{ flow out)}$$

$$\forall (u,v) \in E, \qquad x_{u,v} \le c_{u,v}$$

$$x_{u,v} \ge 0$$

Denote F as the maximum flow in this graph.

2) use another LP to compute the minimal distance flow with the constraint that total amount of flow must be F.

Maximize:

$$\sum_{(u,v)\in E} d_{u,v}x_{u,v}$$

Subject to:

$$\forall v \notin \{s,t\}, \sum_{u:(u,v)\in E} x_{u,v} - \sum_{u:(v,u)\in E} x_{v,u} = 0 \text{ (flow in } = \text{ flow out)}$$

$$\sum_{u:(u,t)\in E} x_{u,t} = -F \text{ (out flow from } t)$$

$$\sum_{u:(s,u)\in E} x_{s,u} = F \text{ (in flow to } s)$$

$$\forall (u,v) \in E, \quad x_{u,v} \leq c_{u,v}$$

$$x_{u,v} \geq 0$$

Solving Shortest Path by LP

Given a directed graph $G = (E, V)$,
denote $d_{u,v}$ as distance of edge $u \to v$

To find the shortest distance from some node s to t

Can be viewed as sending one unit of flow from s to t with minimal total distance.

Solving Shortest Path by LP

Can be viewed as sending one unit of flow from s to t.
Let $x_{u,v}$ be the amount of flow in edge $u \to v$.

Minimize:

$$\sum_{(u,v)\in E} d_{u,v} x_{u,v}$$

Subject to:

$$\forall v \notin \{s,t\}, \sum_{u:(u,v)\in E} x_{u,v} - \sum_{u:(v,u)\in E} x_{v,u} = 0 \text{ (flow in } = \text{ flow out)}$$

$$\sum_{u:(u,t)\in E} x_{u,t} = -1 \text{ (out flow from } t)$$

$$\sum_{u:(s,u)\in E} x_{s,u} = 1 \text{ (in flow to } s)$$

$$\boldsymbol{x} \geq 0$$

Solving Dual Program

Introduce dual variable y_i for each constraints:

Minimize:

$$\sum_{(u,v)\in E} d_{u,v} x_{u,v}$$

Subject to:

$$\forall v \notin \{s,t\}, \sum_{u:(u,v)\in E} x_{u,v} - \sum_{u:(v,u)\in E} x_{v,u} = 0 \qquad y_v$$

$$\sum_{u:(u,t)\in E} x_{u,t} = -1 \qquad y_t$$

$$\sum_{u:(s,u)\in E} x_{s,u} = 1 \qquad y_s$$

$$\boldsymbol{x} \geq 0$$

Dual Program

Maximize:

$$y_t - y_s$$

Subject to:

$$y_v - y_u \leq d_{uv} \quad \text{for all } u, v \in E$$

Dual Program

Maximize:

$$y_t - y_s$$

Subject to:

$$y_v - y_u \leq d_{uv} \quad \text{for all } u, v \in E$$

By strong duality, the optimal value of this dual program is the shortest distance from s to t.

Dual Program

Maximize:

$$y_t - y_s$$

Subject to:

$$y_v - y_u \leq d_{uv} \quad \text{for all } u, v \in E$$

Is this form familiar?

This is the special class of inequality system solved by Bellman Ford Algorithm in Problem 5.1 in HW3.

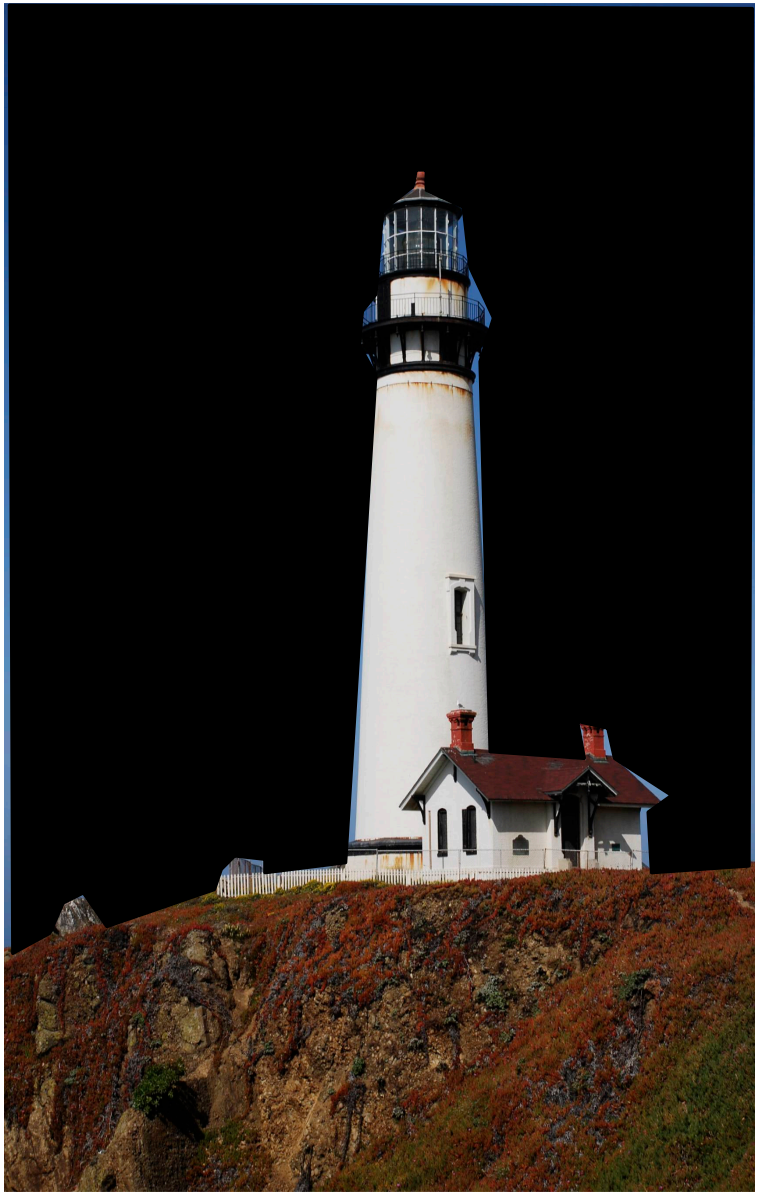
Image Segmentation

- Given an image, what is foreground and what is background?
- E.g. Hockey puck against the ice, football against the field, missiles against the sky.
- If (x, y) is a foreground pixel, then nearby pixels are also likely foreground.

Slides credit to Carl Kingsford (CMU)

Image Segmentation

- Given an image, what is foreground and what is background?
- E.g. Hockey puck against the ice, football against the field, missiles against the sky.
- If (x, y) is a foreground pixel, then nearby pixels are also likely foreground.



Domain Knowledge

Different applications will have different rules for identifying foreground and background pixels, e.g.:

If the pixel is blue, it is more likely to be the background.

Abstract those rules away, and suppose we have 2 nonnegative numbers for each pixel i:

1. a_i = likelihood that pixel i is in foreground.
2. b_i = likelihood that pixel i is in background.

How these values are generated depends on the application.

Clumping

All else equal, if $a_i > b_i$, then we should make i a foreground pixel.

What if it is just a person in blue shirt?

But: we noted that foreground pixels tend to be near one another, and background pixels tend to be near one another.

To represent neighboring pixels, we encode an image by an undirected graph $G = (V, E)$.

- V contains a vertex for each pixel
- E contains an edge between pixels i and j if i and j neighbor each other.

Image Graph Example

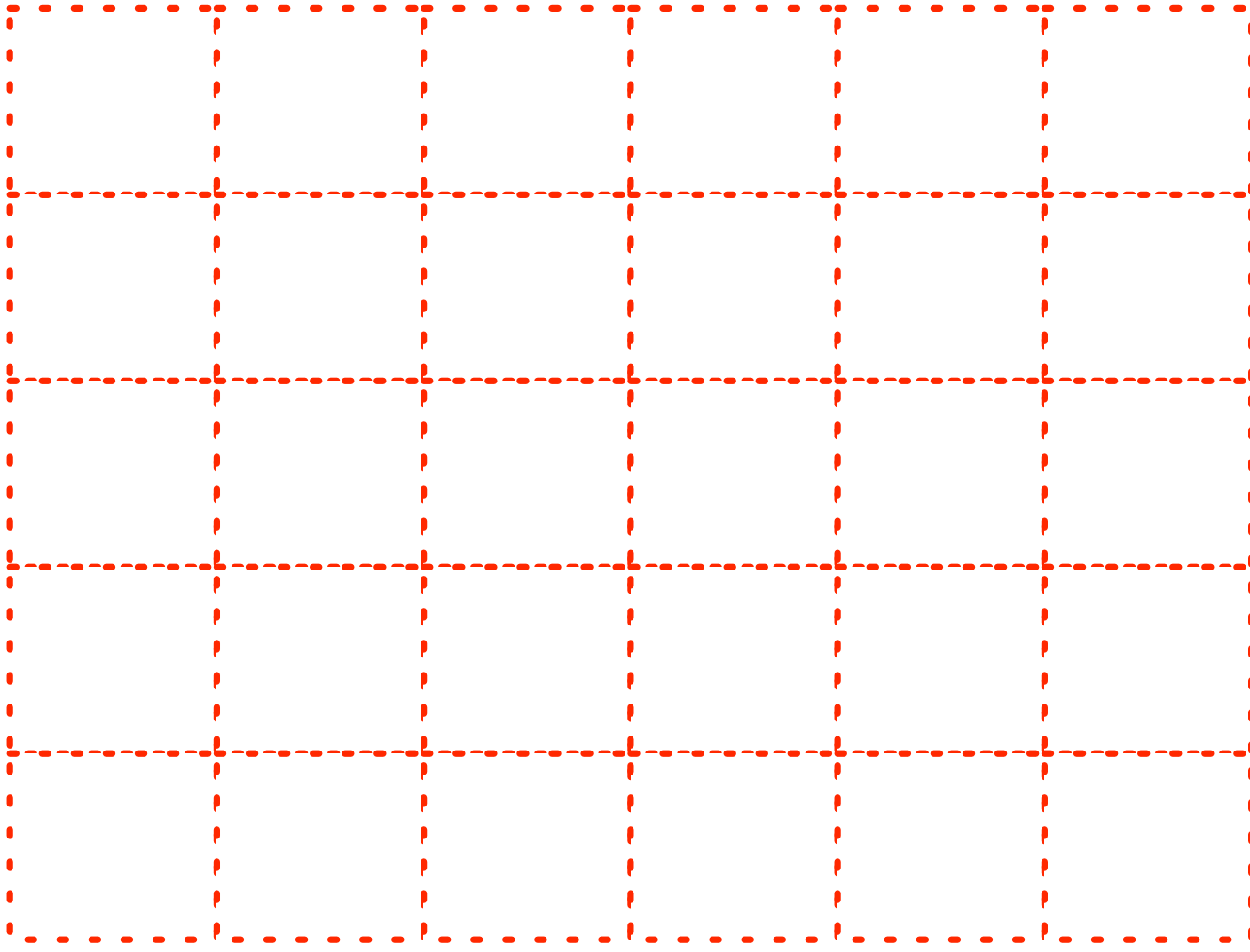


Image Graph Example

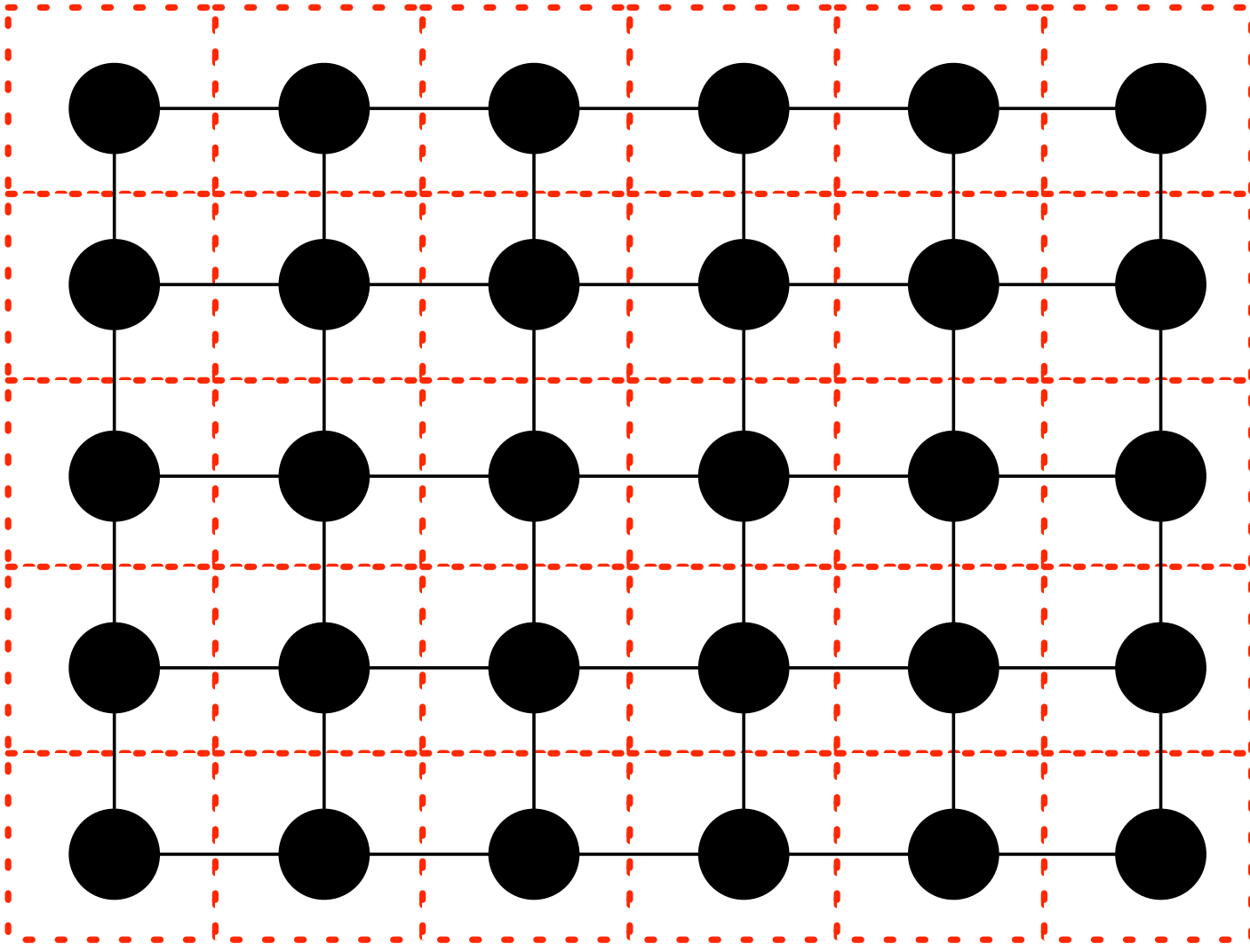


Image Segmentation Problem

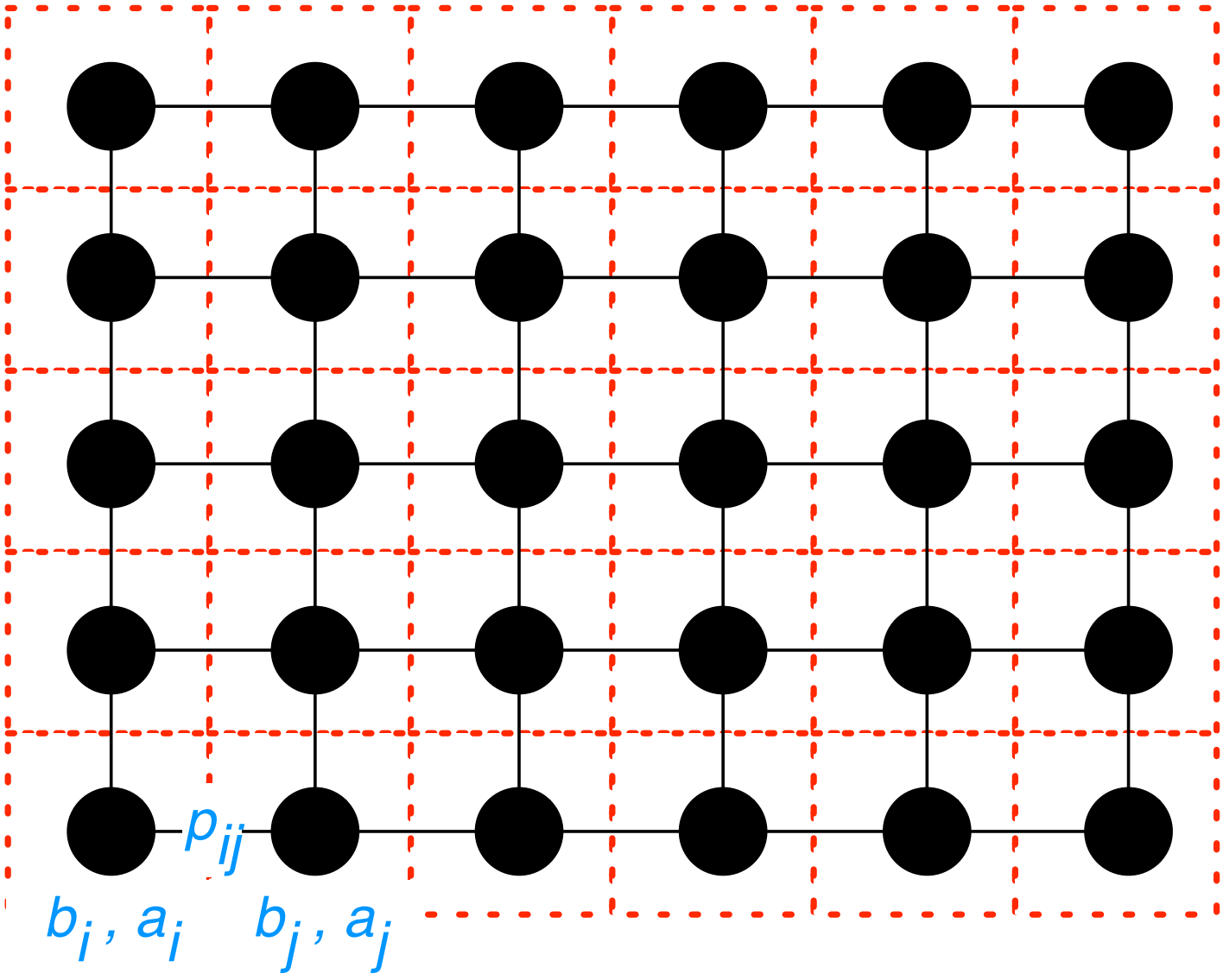
We add new parameters to model separating neighboring pixels. For every neighboring pair of pixels $\{i, j\}$, we have a parameter:

3. p_{ij} = the penalty for putting one of i, j in foreground and the other in the background.

Image Segmentation Problem Partition the set of pixels into two sets A and B to maximize:

$$q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ i,j \text{ sep}}} p_{ij} .$$

Image Graph Example



Min Cut?

Image Segmentation: Partition the vertices of the image graph into 2 sets A, B to maximize $q(A, B)$.

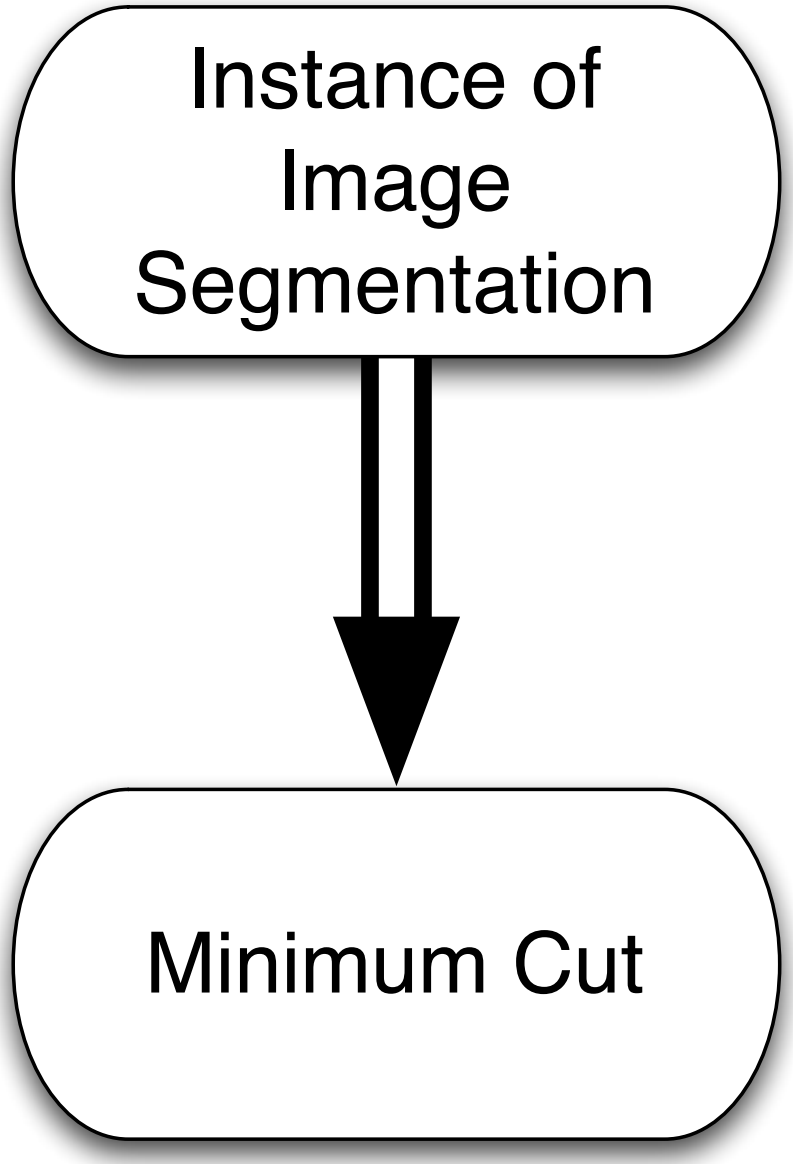
Minimum Cut: Partition the vertices of the a directed graph into 2 sets A, B, with $s \in A$, $t \in B$, to minimize weight of edges crossing from A to B.

Seem similar, but with some differences:

- Maximization vs. minimization
- Image segmentation has no source or sink
- Image segmentation has a more complicated objective function $q(A, B)$ with weights on the nodes
- Undirected vs. directed

Reduction!

- Given an instance of Image Segmentation,
- Create an instance of Minimum Capacity Cut.
- Where the minimum cut can trivially be used to find the solution to the image segmentation.



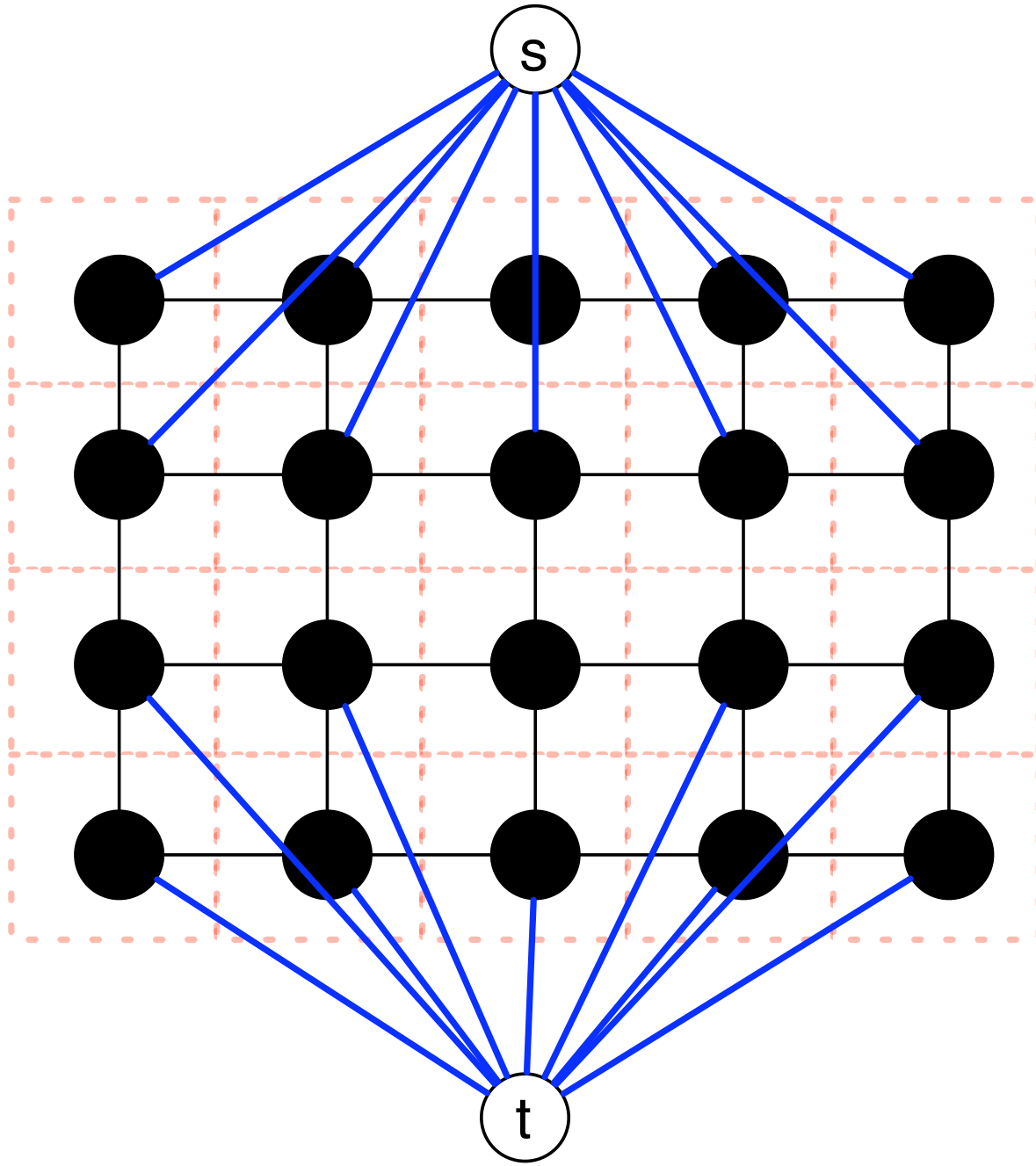
Missing Source and Sink?

We add:

- a source s with an edge (s, u) for every vertex u.
- a sink t with an edge (u, t) for every vertex u.

We will see: s will represent the foreground, and t will represent the background.

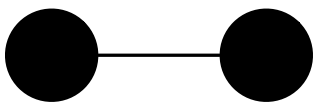
Source and Sink Example



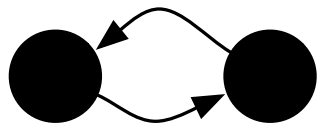
Directed Edges

We convert the currently undirected graph into a directed graph:

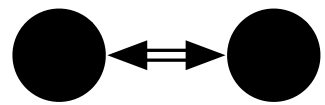
- Edges adjacent to s are directed so they leave s.
- Edges adjacent to t are directed so they enter t.
- All other edges are replaced by 2, anti-parallel edges:



A single undirected edge becomes...

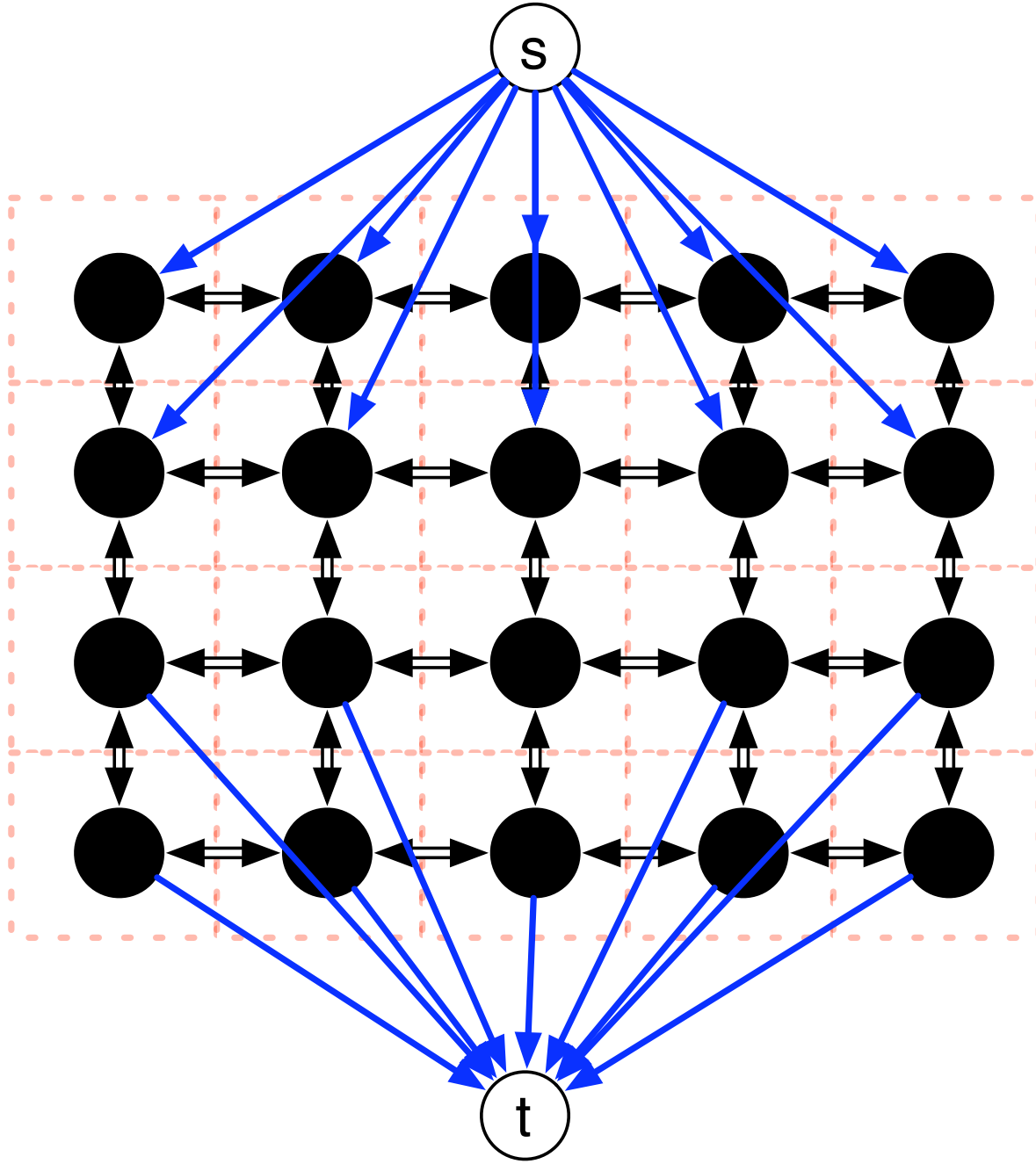


2 anti-parallel edges



Sometimes draw this way to save space

Directed Edges Example



What have we done?

Given an image graph $G = (V, E)$ where V represent the pixels and E connect neighboring pixels, we produced a new graph $G' = (V', E')$ by:

1. $V' = V \cup \{s, t\} \leftarrow$ added source and sink.

2. E' edges from s to every pixel and from every pixel to t plus edges (i, j) and (j, i) between each pair of pixels.

Last issue: how do we handle the parameters a_i, b_i, p_{ij} and minimization vs. maximization?

Maximization vs. minimization

Let $Q = \sum_i (a_i + b_i)$.

Note that:

$$\sum_{i \in A} a_i \; + \; \sum_{j \in B} b_j = Q - \sum_{i \in A} b_i \; - \; \sum_{j \in B} a_j$$

Our old objective function was to maximize:

$$q(A, B) = \sum_{i \in A} a_i \; + \; \sum_{j \in B} b_j \; - \; \sum_{\substack{(i,j) \in E \\ i,j \text{ sep}}} p_{ij}$$

By above, this equals:

$$q(A, B) = Q - \sum_{i \in A} b_i \; - \; \sum_{j \in B} a_j \; - \; \sum_{\substack{(i,j) \in E \\ i,j \text{ sep}}} p_{ij}$$

Maximization vs. Minimization

We want to maximize:

$$q(A,B) = Q - \sum_{i \in A} b_i \; - \; \sum_{j \in B} a_j \; - \; \sum_{\substack{(i,j) \in E \\ i,j \text{ sep}}} p_{ij}$$

This is the same as minimizing:

$$q'(A,B) = \sum_{i \in A} b_i \; + \; \sum_{j \in B} a_j \; + \; \sum_{\substack{(i,j) \in E \\ i,j \text{ sep}}} p_{ij}$$

Parameters

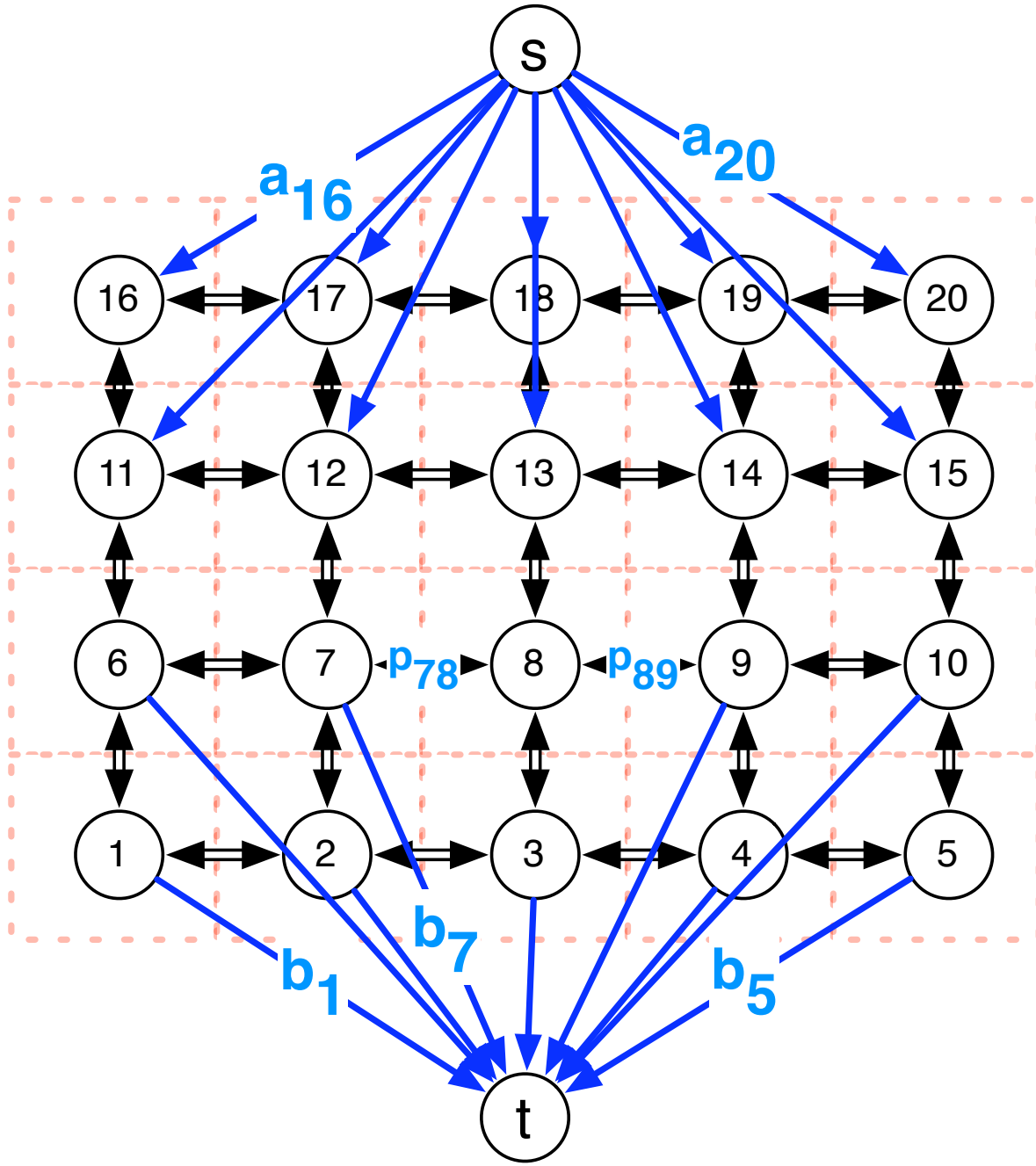
We use the parameters a_i, b_i and p_{ij} as weights on the various edges:

- Edges between two pixels i and j get weight p_{ij}.
- Edge (s, i) gets weight a_i.
- Edge (j, t) get weight b_j.

The intuition here is that the capacity of an $s - t$ cut (A, B) will equal the the quantity we are trying to minimize. (We'll see why.)

Therefore, if we find the min cut, we will find the best partition into foreground and background.

Weights Example



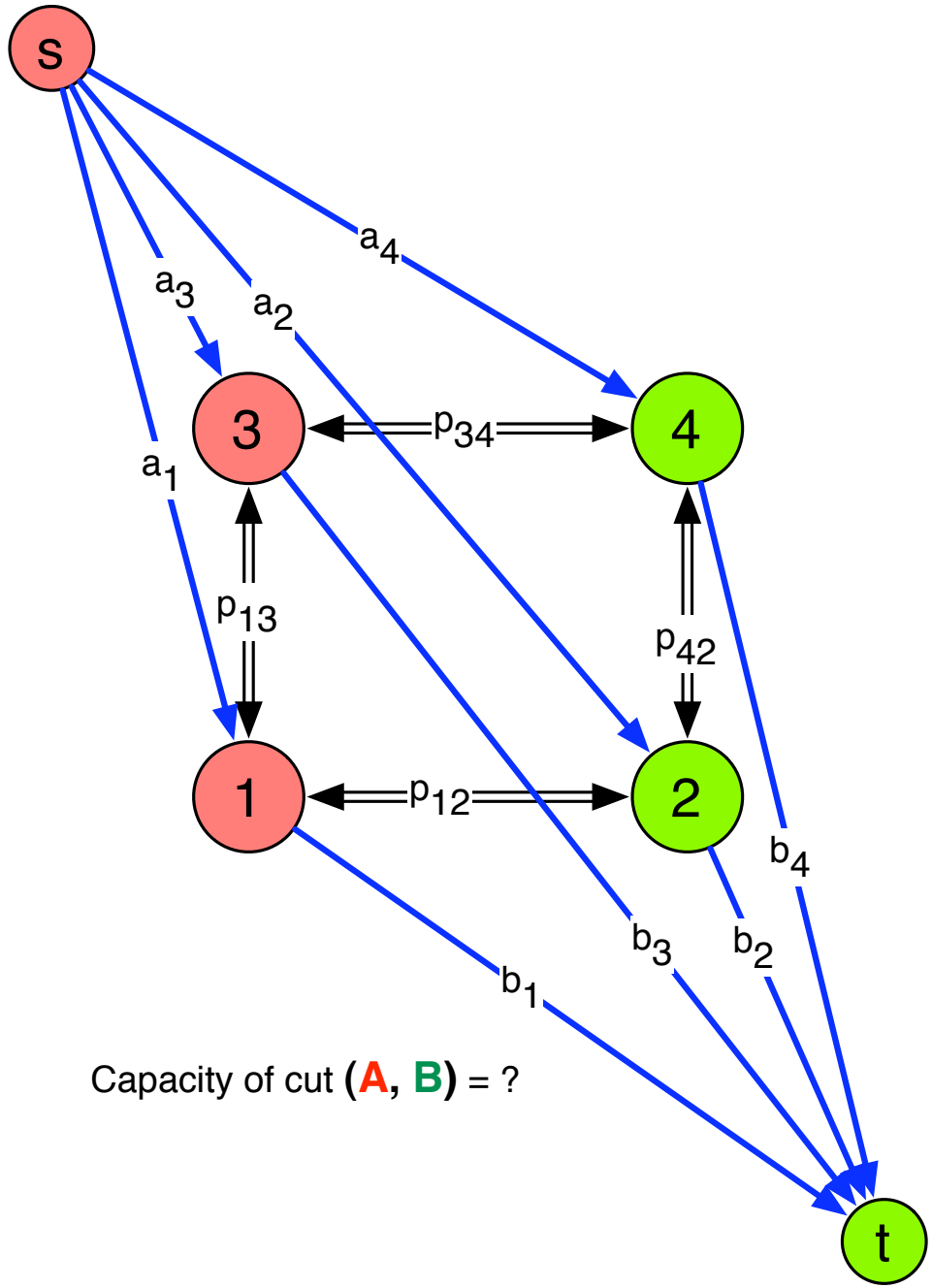
Capacity of a cut = quality of partition

We've designed the graph so that the capacity of an s-t cut (A,B) equals the quality of the partition defined by taking:

- A to be the set of foreground pixels (plus s)
- B to be the set of background pixels (plus t)

Why is this?

This is why:



Capacity of cut **(A, B)** = ?

Cut Edges

$$\text{minimize } q'(A,B) = \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{\substack{(i,j) \in E \\ i,j \text{ sep}}} p_{ij}$$

The edges of any cut (A, B) of our graph can be divided into 3 groups:

- Edges (i, t), where $i \in A$: this edge contributes b_i to the capacity (and putting i into the foreground costs us b_i).
- Edges (s, j), where $j \in B$: this edge contributes a_j to the capacity (and putting j into the background costs us a_j).
- Edges (i, j), where $i \in A, j \in B$: contributes p_{ij} to the capacity (and separating i and j costs us p_{ij}).

Summary

We've shown that for a cut (A, B) in G':

$$\text{capacity}(A, B) = \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{\substack{(i,j) \in E \\ i,\, j \text{ sep}}} p_{ij} = q'(A, B)$$

Hence, finding the minimum capacity cut in our modified graph gives us the partition into foreground A and background B that maximizes $q(A, B)$.

We can find that best partition using a minimum-cut algorithm on the modified graph G' and deleting s and t from the cut it returns.

Approximation Algorithm for Weighted Max-Cut

- We learned two approximation algorithms for Max-Cut in lecture 20.
  - Greedy based : ½ - approximation
  - Randomization based : ½ - approximation

Approximation Algorithm for Weighted Max-Cut

- We learned two approximation algorithms for Max-Cut in lecture 20.
  - Greedy based : ½ - approximation
  - Randomization based : ½ - approximation
- The best known algorithm has a 0.878 - approximation ratio, based on semidefinite programming.

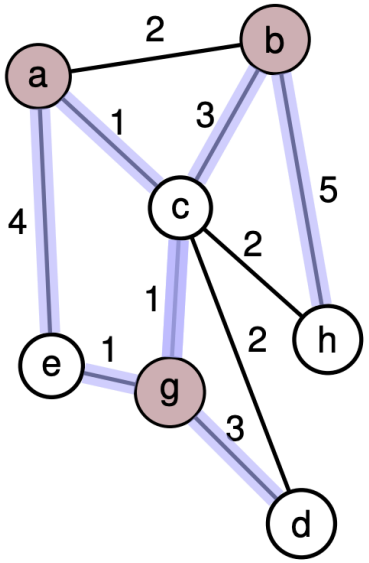
Weighted Max-Cut Problem

Weighted MAX-CUT: Every edge $e \in E$ has a non-negative weight $w(e)$

MAX-CUT Problem

- Given: Undirected graph $G = (V, E)$
- Goal: Find a subset $S \subseteq V$ such that $|E(S, V \setminus S)|$ is maximized.

Weighted MAX-CUT: Maximize the weights of edges crossing the cut, i.e., maximize $w(S) := \sum_{\{u,v\} \in E(S, V \setminus S)} w(\{u, v\})$



$S = \{a, b, g\}$

$w(S) = 18$

Applications:

- cluster analysis
- VLSI design

Quadratic formulation for Max-Cut

Quadratic program

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(i,j) \in E} w_{i,j} \cdot (1 - y_i y_j) \\ \text{subject to} \quad & y_i \in \{-1, +1\}, \qquad i = 1, \ldots, n. \end{aligned}$$

Quadratic formulation for Max-Cut

Quadratic program

$$\begin{aligned} &\text{maximize} && \frac{1}{2} \sum_{(i,j) \in E} w_{i,j} \cdot (1 - y_i y_j) \\ &\text{subject to} && y_i \in \{-1, +1\}, \qquad i = 1, \ldots, n. \end{aligned}$$

➢ This is an integer quadratic programming (IQP) problem.

Quadratic formulation for Max-Cut

Quadratic program

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(i,j) \in E} w_{i,j} \cdot (1 - y_i y_j) \\ \text{subject to} \quad & y_i \in \{-1, +1\}, \qquad i = 1, \ldots, n. \end{aligned}$$

- This is an integer quadratic programming (IQP) problem.
- Vertices labeled by 1,2,…,n and edge weights expressed by $w_{i,j}$.

Quadratic formulation for Max-Cut

Quadratic program

$$\begin{aligned} &\text{maximize} && \frac{1}{2}\sum_{(i,j)\in E} w_{i,j}\cdot(1-y_i y_j) \\ &\text{subject to} && y_i \in \{-1,+1\}, \qquad i=1,\ldots,n. \end{aligned}$$

- This is an integer quadratic programming (IQP) problem.
- Vertices labeled by 1,2,…,n and edge weights expressed by $w_{i,j}$.
- It exactly models the weighted Max-Cut problem.

Vector Programming Relaxation (VPR)

Vector Programming Relaxation

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(i,j) \in E} w_{i,j} \cdot (1 - v_i v_j) \\ \text{subject to} \quad & v_i \cdot v_i = 1 \qquad i = 1, \ldots, n. \\ & v_i \in \mathbb{R}^n \end{aligned}$$

Vector Programming Relaxation (VPR)

Vector Programming Relaxation

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(i,j) \in E} w_{i,j} \cdot (1 - v_i v_j) \\ \text{subject to} \quad & v_i \cdot v_i = 1 \qquad i = 1, \ldots, n. \\ & v_i \in \mathbb{R}^n \end{aligned}$$

➢ The feasible set of the relaxed problem contains the feasible set of the original problem.

Vector Programming Relaxation (VPR)

Vector Programming Relaxation

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(i,j) \in E} w_{i,j} \cdot (1 - v_i v_j) \\ \text{subject to} \quad & v_i \cdot v_i = 1 \qquad i = 1, \ldots, n. \\ & v_i \in \mathbb{R}^n \end{aligned}$$

- The feasible set of the relaxed problem contains the feasible set of the original problem.
- Because any solution of the original IQP can be recovered by setting $v_i = (y_i, 0, 0, \ldots, 0)$.

Positive Semidefinite Matrices

Definition

A matrix $A \in \mathbb{R}^{n \times n}$ is positive semidefinite iff for all $y \in \mathbb{R}^n$,

$$y^T \cdot A \cdot y \geq 0.$$

Positive Semidefinite Matrices

Definition

A matrix $A \in \mathbb{R}^{n \times n}$ is positive semidefinite iff for all $y \in \mathbb{R}^n$,

$$y^T \cdot A \cdot y \geq 0.$$

Theorem 1 : Matrix A is symmetric and positive semidefinite if and only if there exists an $n \times n$ matrix B such that $B^T \cdot B = A$.

Positive Semidefinite Matrices

Definition

A matrix $A \in \mathbb{R}^{n \times n}$ is positive semidefinite iff for all $y \in \mathbb{R}^n$,

$$y^T \cdot A \cdot y \geq 0.$$

Theorem 1 : Matrix A is symmetric and positive semidefinite if and only if there exists an $n \times n$ matrix B such that $B^T \cdot B = A$.

➢ If A is symmetric and positive semidefinite, then the matrix B can be computed in polynomial time (using Cholesky-decomposition).

Reformulate the IQP as a Semidefinite Program(SDP)

Semidefinite Program

maximize $\frac{1}{2} \sum_{(i,j) \in E} w_{i,j} \cdot (1 - a_{i,j})$

subject to $A = (a_{i,j})$ is symmetric and positive definite,

and $a_{i,i} = 1$ for all $i = 1, \ldots, n$

Reformulate the IQP as a Semidefinite Program(SDP)

Semidefinite Program

maximize	$\frac{1}{2} \sum_{(i,j) \in E} w_{i,j} \cdot (1 - a_{i,j})$
subject to	$A = (a_{i,j})$ is symmetric and positive definite,
	and $a_{i,i} = 1$ for all $i = 1, \ldots, n$

- Introduce n^2 variables $a_{i,j} = v_i \cdot v_j$, which give rise to the matrix A.

Reformulate the IQP as a Semidefinite Program(SDP)

Semidefinite Program

$$\begin{aligned} &\text{maximize} && \frac{1}{2}\sum_{(i,j)\in E} w_{i,j}\cdot(1-a_{i,j}) \\ &\text{subject to} && A=(a_{i,j}) \text{ is symmetric and positive definite,} \\ & && \text{and } a_{i,i}=1 \text{ for all } i=1,\ldots,n \end{aligned}$$

- Introduce n^2 variables $a_{i,j} = v_i \cdot v_j$, which give rise to the matrix A.
- If V is the matrix given by $(v_1, v_2, \cdots, v_n)$, then $A = V^T \cdot V$ is symmetric and positive semidefinite.

Reformulate the IQP as a Semidefinite Program(SDP)

Semidefinite Program

maximize	$\frac{1}{2}\sum_{(i,j)\in E} w_{i,j}\cdot(1-a_{i,j})$
subject to	$A=(a_{i,j})$ is symmetric and positive definite,
	and $a_{i,i}=1$ for all $i=1,\ldots,n$

- Introduce n^2 variables $a_{i,j}=v_i\cdot v_j$, which give rise to the matrix A.
- If V is the matrix given by $(v_1, v_2, \cdots, v_n)$, then $A=V^T\cdot V$ is symmetric and positive semidefinite.
- If A is symmetric and positive semidefinite, there must exists a V with $A=V^T\cdot V$ which satisfies the constraint in VPR.

Semidefinite Program(SDP)

- The standard form of SDP

$$SDP: \quad \text{minimize} \quad C \bullet X$$

$$\text{s.t.} \quad A_i \bullet X = b_i \quad , i = 1, \ldots, m,$$

$$X \succeq 0,$$

Semidefinite Program(SDP)

➢ The standard form of SDP

$$SDP: \quad \text{minimize} \quad C \bullet X$$

$$\text{s.t.} \qquad A_i \bullet X = b_i \quad , i = 1, \ldots, m,$$

$$X \succeq 0,$$

➢ The variable X is an $n \times n$ matrix, $\cdot$ is entry-wise product, and $X \succeq 0$ means X is positive semidefinite.

Semidefinite Program(SDP)

➢ The standard form of SDP

$$SDP: \quad \text{minimize} \quad C \bullet X$$

$$\text{s.t.} \qquad A_i \bullet X = b_i \quad , i = 1, \ldots, m,$$

$$X \succeq 0,$$

➢ The variable X is an $n \times n$ matrix, $\cdot$ is entry-wise product, and $X \succeq 0$ means X is positive semidefinite.

➢ SDP is a linear program (LP) with an additional constraint $X \succeq 0$!

Semidefinite Program(SDP)

➢ The standard form of SDP

$$SDP: \quad \text{minimize} \quad C \bullet X$$

$$\text{s.t.} \qquad A_i \bullet X = b_i \quad , i = 1, \ldots, m,$$

$$X \succeq 0,$$

➢ The variable X is an $n \times n$ matrix, $\cdot$ is entry-wise product, and $X \succeq 0$ means X is positive semidefinite.

➢ SDP is a linear program (LP) with an additional constraint $X \succeq 0$!

➢ SDP can be solved in polynomial time (using interior point method).

Back to the Max-Cut problem

➢ To solve a Max-Cut problem instance, we need three steps:

1. solve the corresponding SDP and obtain a positive semidefinite matrix A.

2. decompose $A = V^T \cdot V$ and get $V = (v_1, v_2, \cdots, v_n)$.

3. recover a solution to the original Max-Cut problem from $(v_1, v_2, \cdots, v_n)$.

Back to the Max-Cut problem

➢ To solve a Max-Cut problem instance, we need three steps:

1. solve the corresponding SDP and obtain a positive semidefinite matrix A.
2. decompose $A = V^T \cdot V$ and get $V = (v_1, v_2, \cdots, v_n)$.
3. recover a solution to the original Max-Cut problem from $(v_1, v_2, \cdots, v_n)$.

➢ Step 1, 2 can be done in polynomial time. What about step 3?

Back to the Max-Cut problem

- To solve a Max-Cut problem instance, we need three steps:

1. solve the corresponding SDP and obtain a positive semidefinite matrix A.
2. decompose $A = V^T \cdot V$ and get $V = (v_1, v_2, \cdots, v_n)$.
3. recover a solution to the original Max-Cut problem from $(v_1, v_2, \cdots, v_n)$.

- Step 1, 2 can be done in polynomial time. What about step 3?
- We can design a simple randomized algorithm.

Recover the cut from $(v_1, v_2, \cdots, v_n)$

➢ The algorithm to recover the cut from $(v_1, v_2, \cdots, v_n)$:

Rounding by a random hyperplane :

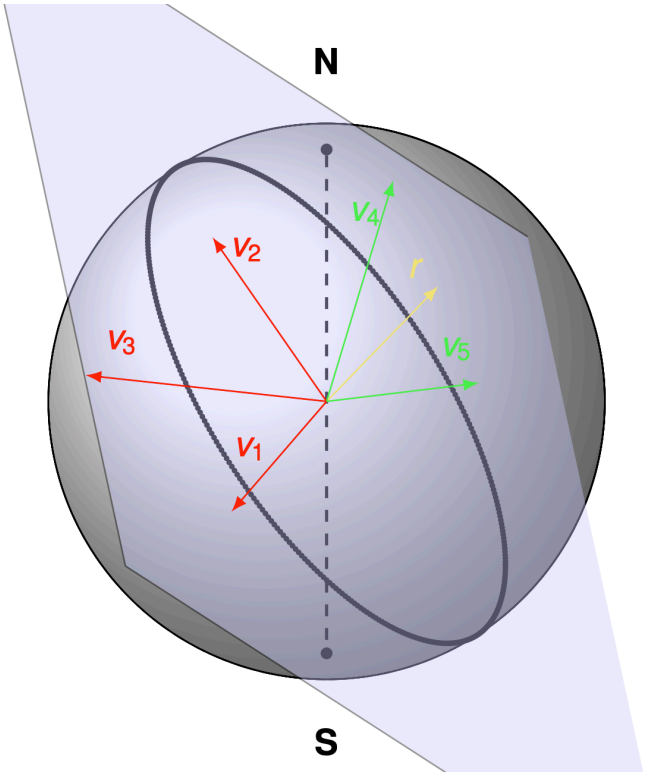
1. Pick a random vector $r = (r_1, r_2, \ldots, r_n)$ by drawing each component from $\mathcal{N}(0, 1)$
2. Put $i \in V$ if $v_i \cdot r \geq 0$ and $i \in V \setminus S$ otherwise

➢ The cut is thus given by V and $V \setminus S$.

When will v_i, v_j be assigned to different cut?

Lemma 1

The probability that two vectors $v_i, v_j \in \mathbb{R}^n$ are separated by the (random) hyperplane given by r equals $\frac{\arccos(v_i \cdot v_j)}{\pi}$.

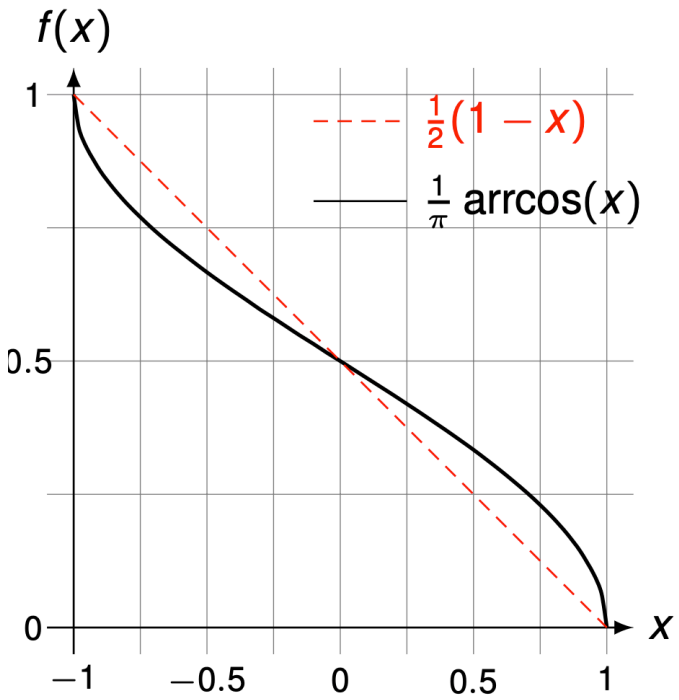


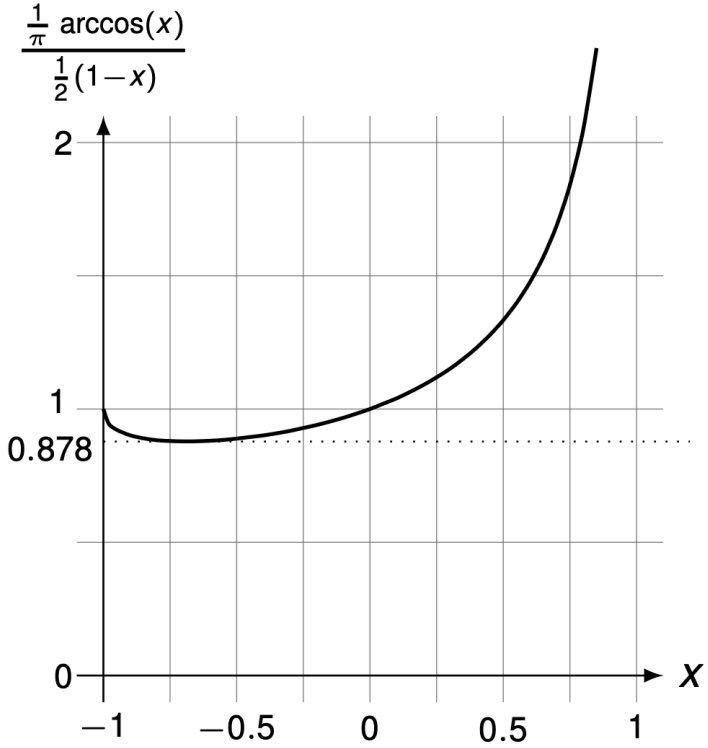
How to estimate the probability

Lemma 2

For any $x \in [-1, 1]$,

$$\frac{1}{\pi}\arccos(x) \geq 0.878 \cdot \frac{1}{2}(1 - x).$$





Estimate the approximation ratio

Theorem (Goemans, Willamson'96)

The algorithm has an approximation ratio of $\frac{1}{0.878} \approx 1.139$.

Proof: Define an indicator variable

$$X_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \text{ are on different sides of the hyperplane} \\ 0 & \text{otherwise.} \end{cases}$$

Hence for the (random) weight of the computed cut,

$$\begin{aligned} \mathbf{E}[\,w(S)\,] &= \mathbf{E}\left[\sum_{\{i,j\}\in E} X_{i,j}\right] \\ &= \sum_{\{i,j\}\in E} \mathbf{E}\left[X_{i,j}\right] \\ &= \sum_{\{i,j\}\in E} w_{i,j} \cdot \mathbf{Pr}\left[\{i,j\} \in E \text{ is in the cut}\right] \\ \text{By Lemma 1} \quad &= \sum_{\{i,j\}\in E} w_{i,j} \cdot \frac{1}{\pi} \arccos(v_i \cdot v_j) \\ \text{By Lemma 2} \quad &\geq 0.878 \cdot \frac{1}{2} \sum_{\{i,j\}\in E} w_{i,j} \cdot (1 - v_i \cdot v_j) = 0.878 \cdot z^* \geq 0.878 \cdot W^*. \end{aligned}$$

□