

# Graph Partitioning

## Introduction

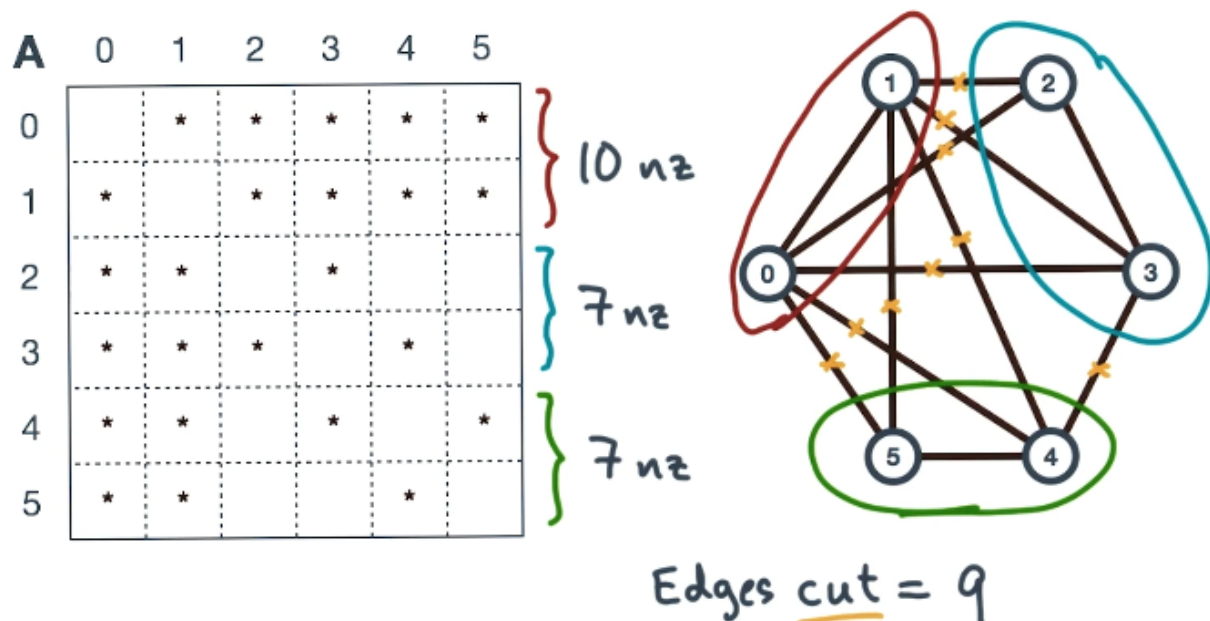
1. Common thread in a distributed memory algorithm: How do you distribute the data?
  - Spectral partitioning: Exploits the connection between graphs and linear algebra
    - Physics-based interpretation based on systems of springs

## The Graph Partitioning Problem

1. Can represent a graph as an adjacency matrix
  - Can partition the matrix across rows, corresponding to a graph (vertex) partition (sparse matrix)
  - Can then do a BFS with a matrix-vector multiply
    - Implies partitioning the vector as well
  - Amount of work corresponds to the number of non-zero entries
    - Ideally, we should balance the work
  - Any time an edge crosses a process boundary, a communication exchange occurs
    - Reduce edge cuts to minimize communication volume
2. Two goals:
  - Balance work across processes
  - Minimize communication between processes
3. Graph partitioning problem:
  - Given  $G = (V, E)$  and number of partitions,  $P$
  - Compute a (vertex) partition  $V = V_0 \cup V_1 \cup \dots \cup V_{p-1}$  such that:
    - $\{V_i\}$  are disjoint  $\Rightarrow \text{intersect}(V_i, V_j) = 0$
    - $\{V_i\}$  are roughly balanced  $\Rightarrow |V_i| \sim |V_j|$
    - Let  $E_{\text{cut}} = \{(u, v) \mid u \text{ in } V_i, v \text{ in } V_j, i \neq j\}$  and minimize  $E_{\text{cut}}$

## Do You Really Want a Graph Partition?

1. Does the graph partitioning problem as formalized above meet the intended goals?
  - No. Despite the number of vertices per partition being the same, the number of non-zero entries can vary, so the work is not evenly distributed

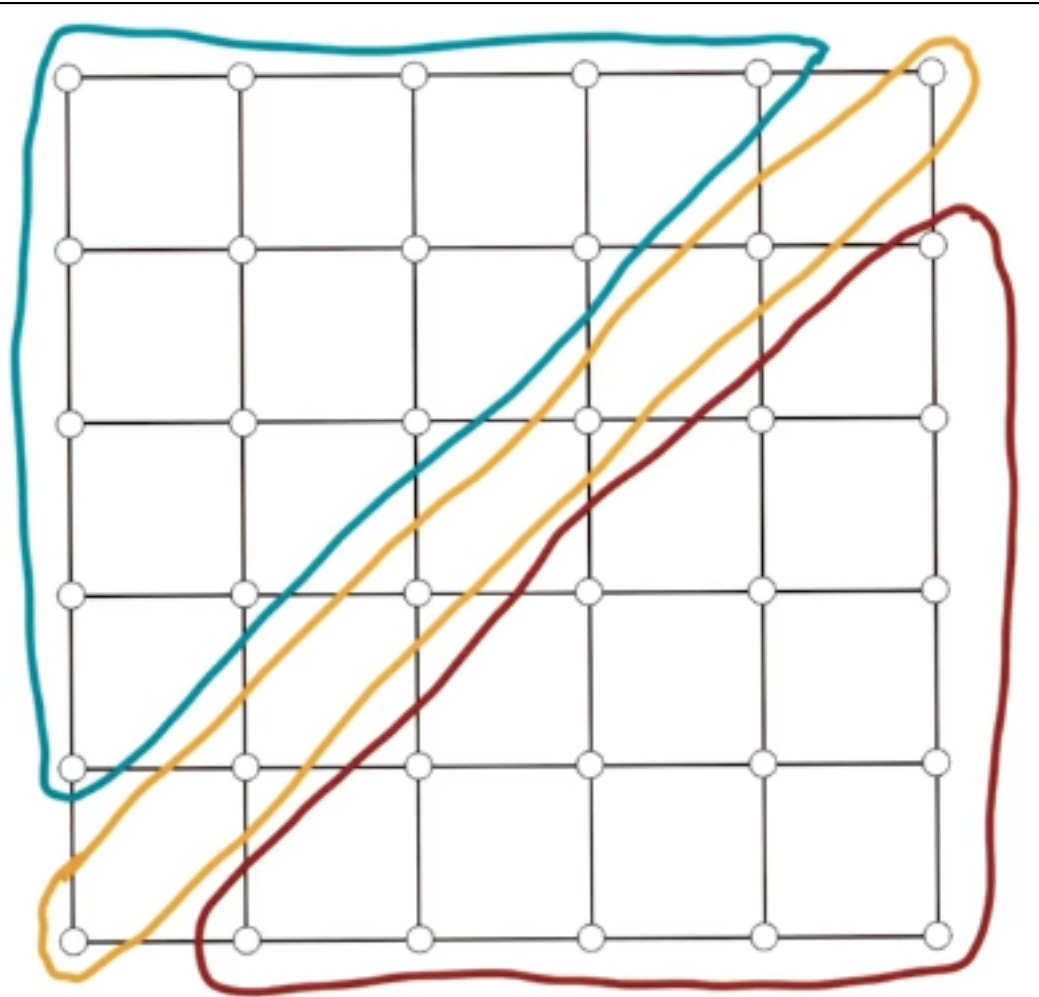


Uneven Distribution of Work

---

## Graph Bisection and Planar Separators

1. Graph partitioning is NP-complete -> Need heuristics!
  - Heuristic: Bisection (divide and conquer)
    - Suppose you want P partitions
    - Start by using any algorithm to divide the graph into 2 partitions
    - Then, divide each half into two more partitions
    - Repeat until the P partitions are obtained
2. Planar graphs
  - Graphs that can be drawn in 2D with no edge crossings
3. Planar separators
  - Theorem: A planar graph  $G = (V, E)$  with  $|V| = n$  vertices has a disjoint partition  $V = A \cup S \cup B$  such that:
    - S separates A and B
    - $|A|, |B| \leq 2n / 3$  (the larger of the two partitions is no more than twice the size of the smaller)
    - $|S| \leq 2 * \sqrt{2n} = O(\sqrt{N})$
    - Lipton and Tarjan 1979
  - Lipton and Tarjan described a polynomial-time algorithm




---

Planar Graph

---

## Partitioning via Breadth First Search

1. How can BFS be used to bisect a graph? Assume the graph is connected
  - Run BFS from any vertex
    - BFS will assign every vertex to a level
    - Levels separate subgraphs
  - Stop when about 1/2 of the vertices have been visited. Assign visited to one partition, unvisited to the other
    - Other criteria are possible
2. However, we wanted to use graph partitioning to distribute our BFS computation, so we can't use BFS to determine the graph partition

## Kernighan Lin - Part 1: No Gain is Pain

1. Kernighan-Lin algorithm is the most well-known heuristic for graph partitioning
  - Given a graph, divide the vertices into two subsets of equal or nearly- equal size
    - $V = V1 \cup V2$ ,  $|V1| = |V2|$
  - Cost of this partition is the number of edges between  $V1$  and  $V2$

- $\text{Cost}(V1, V2) = \text{Number of edges between } V1 \text{ and } V2$
  - Assume you have two evenly-sized subsets of  $V1$  and  $V2$ 
    - $X1$  from  $V1$ ,  $X2$  from  $V2$  with  $|X1| = |X2|$
    - If you swapped these two subsets, the cost will change, but by how much?
2. Formal definitions
- Consider vertex  $a$  in  $V1$  and vertex  $b$  in  $V2$
  - External costs: Number of vertices in the opposite partition
    - $E1(a \text{ in } V1) = \# \text{ edges}(a, b \text{ in } V2)$
    - $E2(b \text{ in } V2) = \# \text{ edges}(b, a \text{ in } V1)$
  - Internal costs: Number of vertices in the same partition
    - $I1(a \text{ in } V1) = \# \text{ edges}(a, i \text{ in } V1)$
    - $I2(b \text{ in } V2) = \# \text{ edges}(b, i \text{ in } V2)$
  - $\text{Cost}(V1, V2) = \text{Cost}(V1 - \{a\}, V2 - \{b\}) + E1(a) + E2(b) - c(a,b)$ 
    - $c$  is a constant to account for an edge between  $a$  and  $b$ . 1 if an edge exists, 0 otherwise
  - Then, consider the cost after swapping  $a$  and  $b$
  - $\text{Cost}(V1, V2) = \text{Cost}(V1 - \{a\}, V2 - \{b\}) + I1(a) + I2(b) + c(a,b)$
  - Change in cost =  $E1(a) + E2(b) - I1(a) - I2(b) - 2c(a,b)$ 
    - Larger change is better  $\Rightarrow$  Larger decrease in cost
  - Change in cost =  $\text{gain}(a \text{ in } V1, b \text{ in } V2)$ 
    - Can be negative if cost increased

## Kernighan Lin Algorithm Quiz

1. Consider the computation of the gain function  $a$  and  $b$ 
  - Assume the following:
    - Every vertex has a partition label;  $O(1)$  access time
    - Maximum degree of any vertex is  $d$
2. What is the sequential running time to compute  $\text{gain}(a, b)$  in terms of  $d$ ,  $n1 = |V1|$ ,  $n2 = |V2|$ 
  - $O(d)$
  - Need to sweep over the adjacent vertices, which is at most  $d$ 
    - Need to check partition label of each vertex, which is a constant-time operation

## Kernighan-Lin Algorithm

1. How do we choose  $X1$  and  $X2$ , the subsets of the partitions?

```

let V = V1 U V2
C = cost(V1, V2)
forall a in V1, do compute E1(a), I1(a)
forall b in V2, do compute E2(b), I2(b)
forall v in V do visited[v] <- false

while !all(visited) do
    choose unmark (a in V1, b in V2) with largest gain(a, b)
    visited[a], visited[b] <- true
    Update all E1, E2, I1, I2 // not actually swapping nodes, just updating costs

// this provides gain(a1, b1), gain(a2, b2), ...
let Gain(j) = sum(gain(a, b))
choose jmax = argmax(Gain(j))
if Gain(jmax) > 0 then
    X1 = {a1, a2, ... ajmax}
    X2 = {b1, b2, ... bjmax}
    Update C <- C - Gain(jmax)
    V1 <- (V1 - X1) U X2 // swap

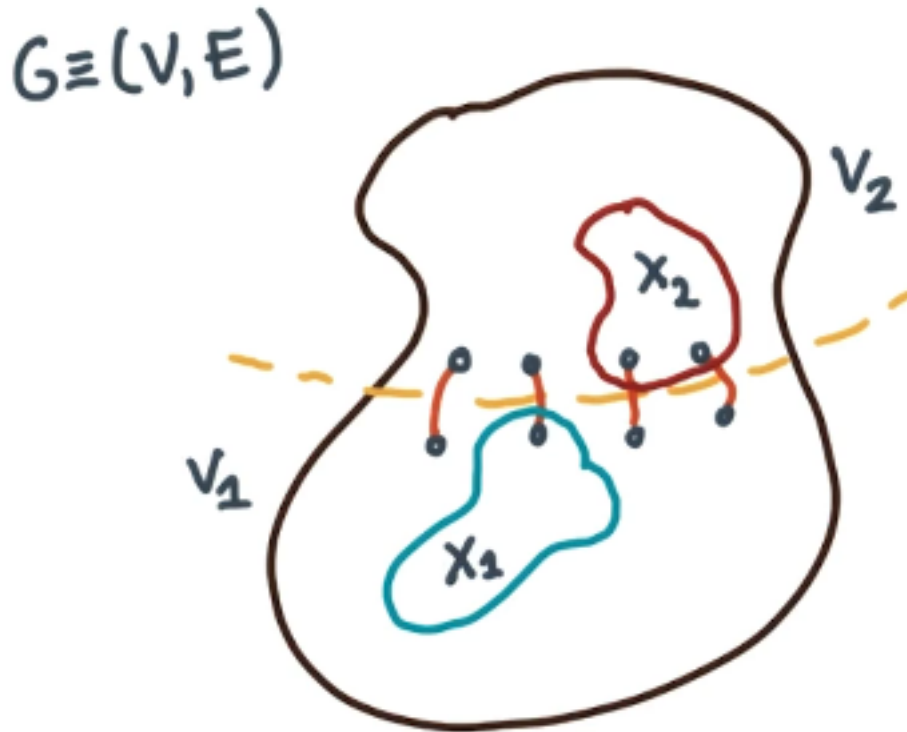
```

```
V2 <- (V2 - X2) U X1 // swap
```

```
// repeat until Gain(jmax) < 0
```

2. Cost:

- Overall running time  $O(|V|^2 * d)$
- Can be reduced to  $O(|E|)$ 
  - Fiduccia and Mattheyses 1982



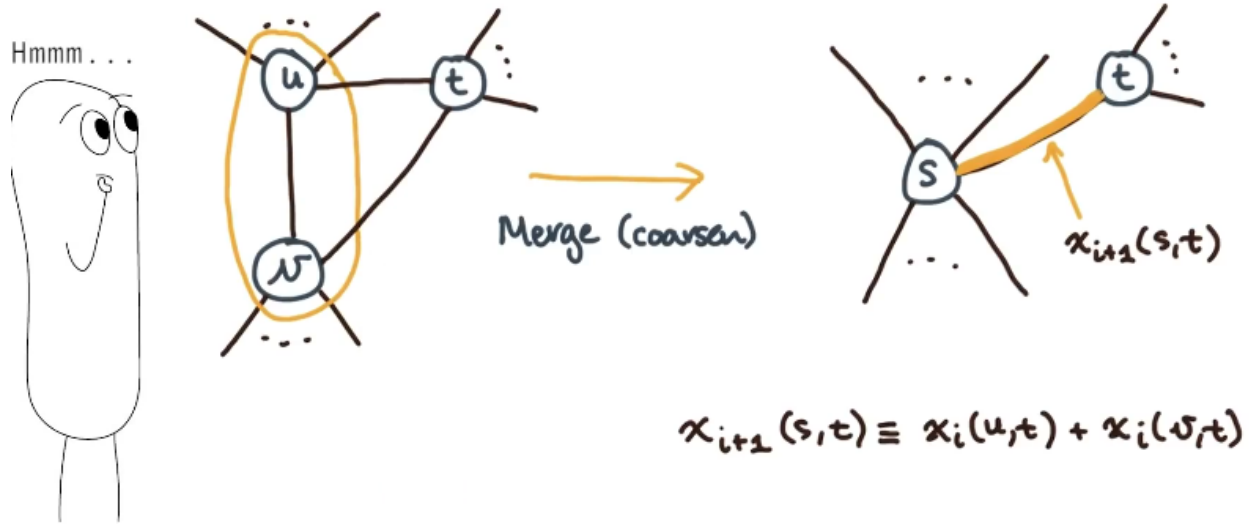
Kernighan-Lin Algorithm

## Graph Coarsening

1. Multi-level graph coarsening: Different partitioning scheme
  - Divide-and-conquer
  - Continually replace graph with smaller graphs (fewer nodes and edges) but still somehow resembles original graph
    - Repeat until the graph is small enough to partition quickly
    - If we've done a good job of preserving the shape of the graph, the split will correspond to a roughly equivalent split in the larger graph
2. How do you actually coarsen a graph?
  - Identify at least one subset of the vertices to collapse
  - Replace subset with single "super" vertex
  - Track the fact that one vertex has replaced many by assigning a weight  $W$  to the vertex, where  $W$  is the number of replaced vertices
  - Track edge weights as well so we can cut edges accurately later on
  - $G_i = (V_i, w_i: V_i \rightarrow R, E_i, x_i: W_i \rightarrow R)$ 
    - $w_i$  and  $x_i$  are functions that map edges and vertices to super edges and vertices
    - Simply sum the contributions from each vertex

$$G_i = (V_i, w_i: V_i \rightarrow \mathbb{R}, E_i, x_i: E_i \rightarrow \mathbb{R})$$

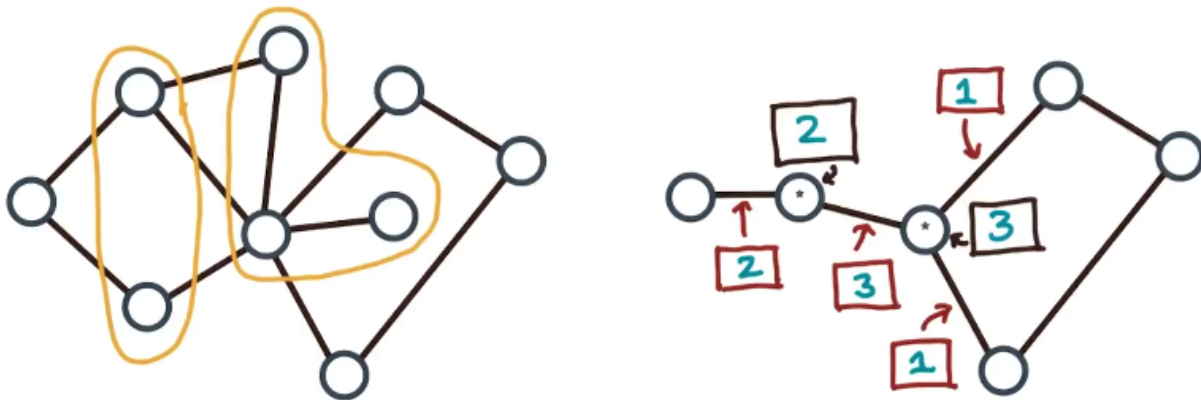
$$G_{i+1}$$



Graph Coarsening Algorithm

### Coarsen Me - Baby

1. Determine the coarsen vertex and edge weights in the following graph:



Graph Coarsening Quiz

### Maximal and Maximum Matchings

1. Matching: A subset of a graph  $G = (V, E)$  is a subset  $E'$  of  $E$  of edges with no common endpoints
  - Independent set, but for edges instead of vertices
2. Maximal matching: A matching is maximal if no more edges may be added
3. Maximum matching: A graph's maximum matching is its largest (most edges or total edge weight)

Def.: A graph's **maximum** matching is its largest (most edges or total edge weight).



---

Maximal (and Maximum) Matching

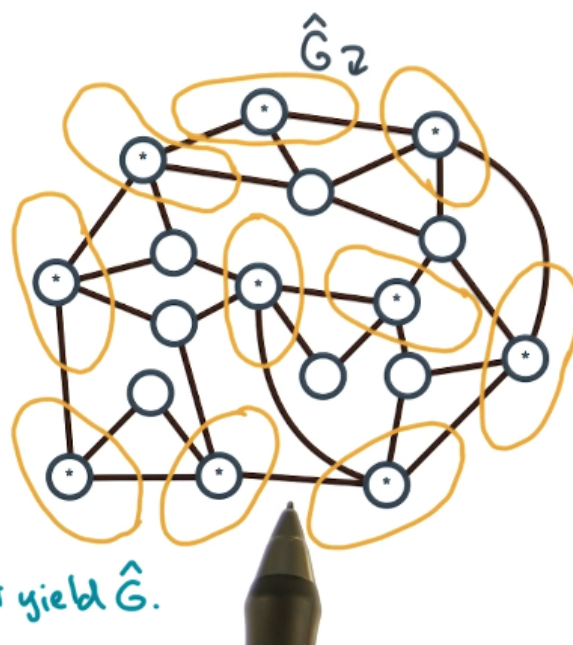
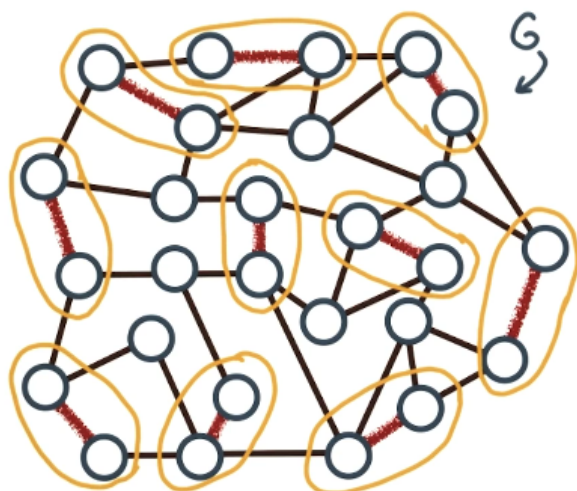
---

### Find the Maximal Matching

1. Mark the edges of  $G$  that yield  $G_{\text{hat}}$

---

### Quiz! Find the Maximal Matching



Your task: Mark the edges of  $G$  that yield  $\hat{G}$ .

---

### Maximal Matching Quiz

---

#### A Fact about Maximal Matchings

1. Consider an initial graph  $G$ . Suppose you find a maximal matching and use it to coarsen. Repeat this process producing a sequence of graphs from 1 to  $k$ . The original graph has  $n$  vertices and the final graph has  $s$  vertices.
2. How large must  $k$  be in terms of  $n$  and  $s$ ?
  - $\log(n/s)$
  - Each level must have at least half of the previous number of vertices, so  $|V_k| = n/2^k \Rightarrow k \geq \log_2(n/s)$

#### Computing a Maximal Matching

1. Pick an unmatched vertex at random
2. Match to one of its unmatched neighbors
  - Could pick randomly
  - Better strategy is to pick the heaviest edge
    - Not a lot of rigorous analysis, but lots of experimental evidence
    - The intuition comes from the fact that picking the heaviest edge should lead to the greatest reduction in edge weight when coarsening the graph

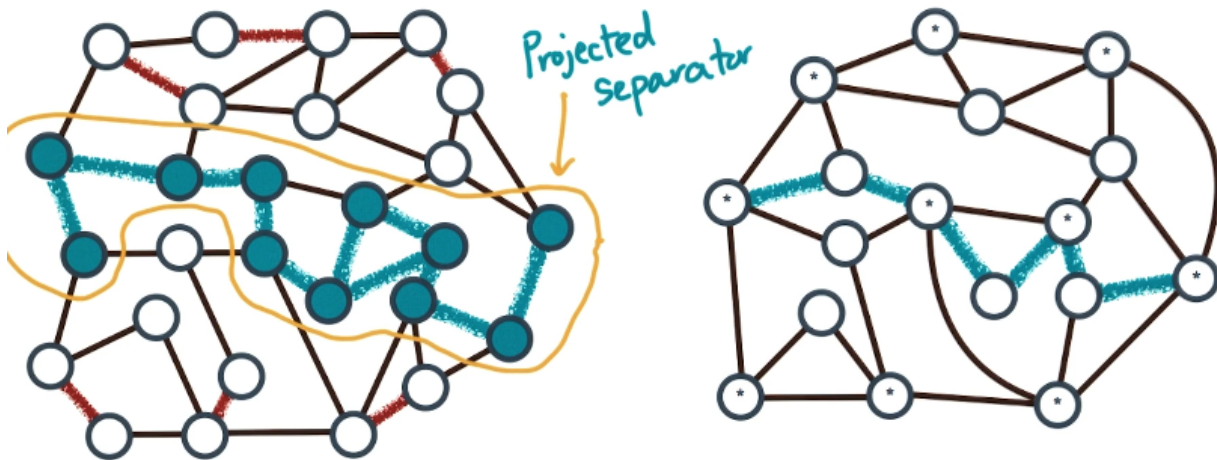
#### Fine-to-Coarse and Back Again

1. Mark the planar separator from the coarsened graph in the original graph
  - Some separator edges map ambiguously if they were merged in the coarsened graph



---

### Quiz! Fine-to-Coarse & Back Again



Your task: Mark the separator in the original (left) graph.

---

Planar Separator in Coarsened Graph

---

### Partition Refinement

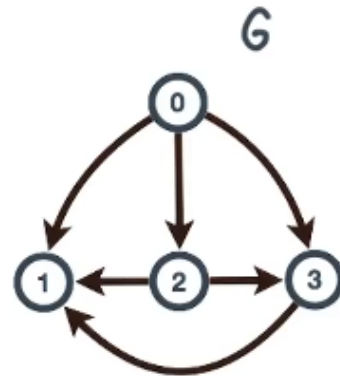
1. A minimum balanced edge cut in a coarsened graph minimizes the balanced edge cut in the next finer graph.
  - False; Coarsening is a heuristic, so it's possible that there's a better cut in the finer graph

### Spectral Partitioning - Part 1: The Graph Laplacian

1. Instead of representing a graph as an adjacency matrix, represent it as an incidence matrix
  - Each row is an edge, each column is a vertex
  - Graph Laplacian,  $L(G) = C' * C$ 
    - Diagonals: Count incident edges (always 1)
    - Off-diagonals: Says edges are adjacent
    - Tells us something about the undirected form of the original graph; we lose the direction information when calculating the Laplacian
  - Graph Laplacian,  $L(G) = C' * C = D - W$ 
    - $D$  is the degree of each vertex (along the diagonal)
    - $W$  is the adjacency matrix marking all the edges

$C = C(G)$ : Incidence matrix

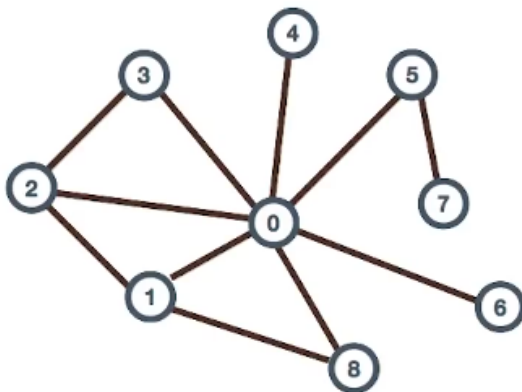
	0	1	2	3	
0	1	-1			$0 \rightarrow 1$
1	1		-1		$0 \rightarrow 2$
2	1			-1	$0 \rightarrow 3$
3		-1	1		$2 \rightarrow 1$
4			1	-1	$2 \rightarrow 3$
5		-1		1	$3 \rightarrow 1$



Incidence Matrix Representation

## Graph Laplacian

1. Compute the graph Laplacian for the following graph
  - Matrix should be symmetric due to the definition of the Laplacian



	0	1	2	3	4	5	6	7	8
0	7	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	3	-1						-1
2	-1	-1	3	-1					
3	-1		-1	2					
4	-1				1				
5	-1					2	-1		
6	-1						1		
7						-1		1	
8	-1	-1							2

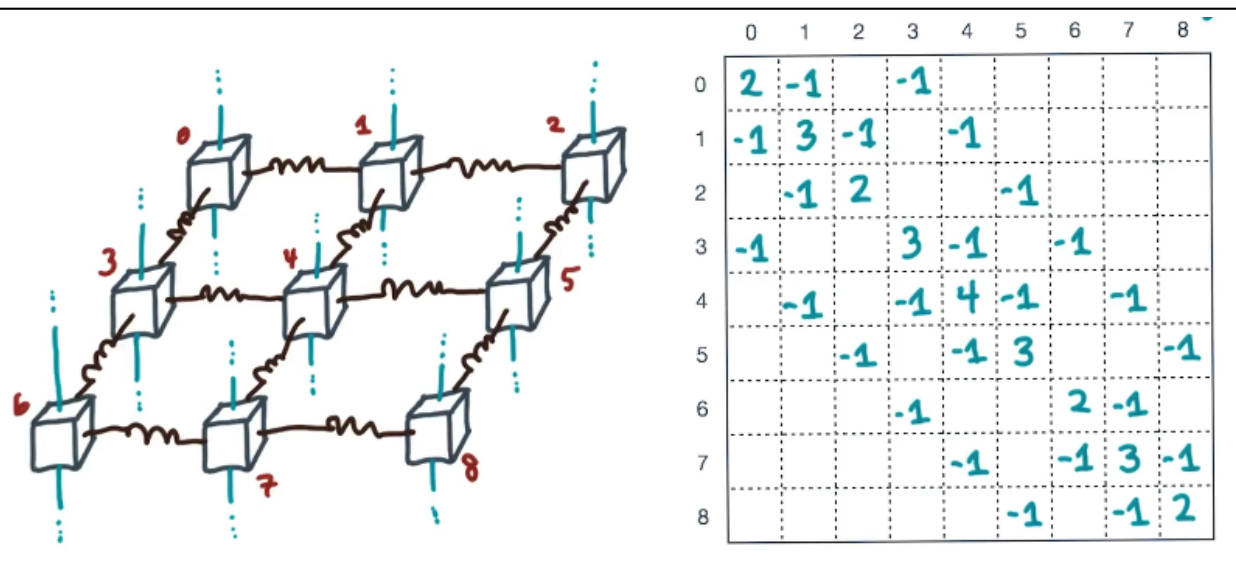
Laplacian Quiz 1

## Spectral Partitioning - Part 2: Springs Fling

1. If we imagine a series of weights connected with springs displaced from a common point  $x$ , it provides some physical intuition for the Laplacian
  - Force on one weight is proportional to the displacement of the adjacent weights
  - This corresponds to the graph Laplacian for a line graph

## A 2D Laplacian

1. What is the graph Laplacian for a group of 9 weights connected in the following way:



Laplacian Quiz 2

## Spectral Partitioning - Part 3: Algebraic Connectivity

1. Factoids

- Laplacian  $L(G)$  is symmetric
- $L(G)$  has real-valued, non-negative eigenvalues and real-valued, orthogonal eigenvectors (not complex)
  - Eigenvalues and eigenvectors are pairs; multiplying  $L(G)$  by its eigenvector gives a scaled version of the eigenvector. The scaling factor is the eigenvalue
  - $L(G) * Q = Q * Y$  where  $Q$  and  $Y$  are matrices
  - Columns of  $Q$  are the eigenvectors and diagonal entries of  $Y$  are the eigenvalues
  - Convention: Assume we can sort the eigenvalues
- $G$  has  $k$  connected components if and only if the  $k$  smallest eigenvalues are identically 0
  - Spectrum of  $L(G)$  tells us something about the connectivity of  $G$
- Let  $V+$  be the number of vertices in the positive section and  $V-$  be the number of vertices in the negative section
  - Define  $x$  such that  $x_i$  is 1 if  $i$  is in  $V+$  and -1 if  $i$  is in  $V-$
  - Then, the number of cut edges =  $1/4 * x' * L(G) * x$

---

$G$

let  $V_+ \equiv \{\text{vertices in "+"}\}$   
 $V_- \equiv \{\text{vertices in "-"}\}$

let  $\vec{x} \equiv \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}$  s.t.  $x_i = \begin{cases} +1 & \text{if } i \in V_+ \\ -1 & \text{if } i \in V_- \end{cases}$

Fact 4) # of **cut edges** =  $\frac{1}{4} \vec{x}^T L(G) \vec{x}$

---

### Spectral Partitioning

### Counting Edge Cuts

#### Quiz! Counting Edge Cuts

$G$

Your task:  
 Fill in the boxes!

$$\begin{aligned}
 \vec{x}^T L(G) \vec{x} &= \sum_{i,j} l_{ij} x_i x_j \\
 &= \sum_{i,j} l_{ij} x_i x_j \quad \leftarrow = l_{ii} x_i^2 \quad \boxed{d_i} \quad \boxed{+1} \\
 &+ \sum_{\substack{i,j \in V_+ \\ i \neq j}} l_{ij} x_i x_j \quad \boxed{-1} \quad \boxed{1} \quad \boxed{1} + \sum_{\substack{i,j \in V_- \\ i \neq j}} l_{ij} x_i x_j \quad \boxed{-1} \quad \boxed{1} \quad \boxed{1} \\
 &+ \sum_{\substack{i \in V_+ \\ j \in V_-}} l_{ij} x_i x_j \quad \boxed{-1} \quad \boxed{1} \quad \boxed{-1} + \sum_{\substack{i \in V_- \\ j \in V_+}} l_{ij} x_i x_j \quad \boxed{-1} \quad \boxed{-1} \quad \boxed{1}
 \end{aligned}$$

### Laplacian Quiz 3

### Spectral Partitioning - Part 4

1. Start with a graph  $G = (V, E)$
2. Construct its Laplacian  $L(G) = D - W$
3. Suppose we have a partitioning of the vertices  $V = V_+ \cup V_-$
4. Translate this into a partition vector  $\vec{x} = \{+1 \text{ if } i \in V_+, -1 \text{ if } i \in V_-\}$

5. If we want to minimize the cut edges, we need to pick an  $x$  to minimize  $1/4 * x' * L(G) * x$ 
  - Want:  $\min(1/4 * x' * L(G) * x)$ 
    - $\sum(x_i) = 0$  (same number of vertices in each partition)
    - $x_i = +1$  or  $-1$
  - These constraints make the problem NP-complete
    - If we relax the constraint that we assign a  $+1$  or  $-1$  to every vertex, we can use the Courant-Fisher Minimax theorem
    - This says that the vector that minimizes this problem is  $q_1$ , which is the eigenvector corresponding to the second smallest eigenvalue of  $L(G)$
  - Choosing  $x$  to be  $q_1$  gives us a lower bound, but how do we take  $q_1$  and turn it into a partition vector?
    - Based on the spring model, the second smallest eigenvector has some sine-like shape where half the values are positive and half are negative
6. Spectral partitioning algorithm:
  - Create  $L(G)$
  - Compute  $(Y, q_1)$  eigenpair of  $L(G)$
  - Choose  $x(i) = \text{sign}(q_1(i))$
7. Spectral partitioning works very well for planar graphs

## Conclusion

1. Graph partitioning is NP-complete
  - Want algorithms with good heuristics
2. Heuristics:
  - Multilevel partitioning (divide and conquer)
  - Exploiting special structure (planarity)
  - Improvement techniques (Kernighan-Lin)
3. Spectral partitioning: Exploits relationship between graphs and matrices
  - K-L and spectral partitioning are very expensive relative to the motivation, which was a relatively cheap BFS