Graph partitioning for parallel tensor contractions

Aydin Buluc

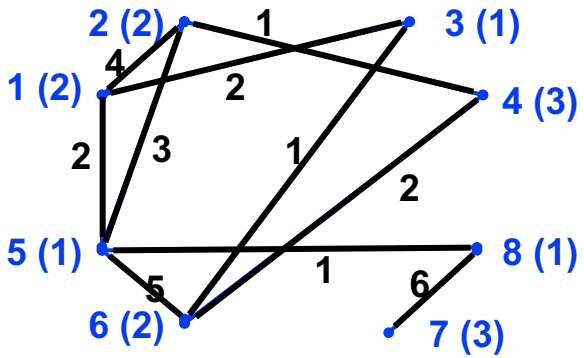
Graph partitioning slide credits:
Umit Catalyurek, James Demmel, John Gilbert, George Slota

https://sites.google.com/lbl.gov/cs267-spr2021/

Tensor contraction slide credits: Jie Chen, Robert Edwards

Definition of Graph Partitioning

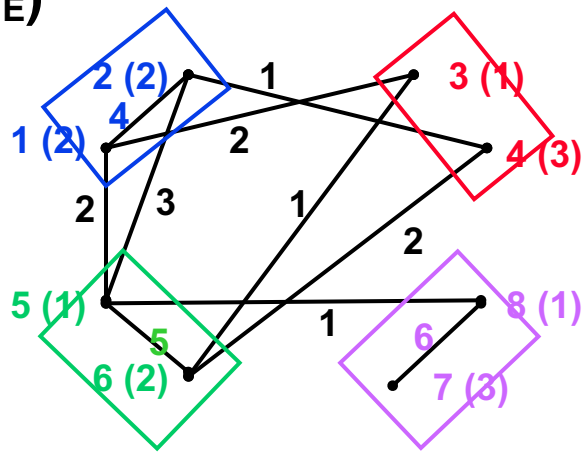
- Given a graph G = (N, E, W_N, W_E)
  - N = nodes (or vertices),
  - W_N = node weights
  - E = edges
  - W_E = edge weights



- Ex: N = {tasks}, W_N = {task costs}, edge (j,k) in E means task j sends W_E(j,k) words to task k
- Choose a partition N = N_1 U N_2 U … U N_P such that
  - The sum of the node weights in each N_j is "about the same"
  - The sum of all edge weights of edges connecting all different pairs N_j and N_k is minimized
- Ex: balance the work load, while minimizing communication
- Special case of N = N_1 U N_2: Graph Bisection

Definition of Graph Partitioning

- Given a graph G = (N, E, W_N, W_E)
  - N = nodes (or vertices),
  - W_N = node weights
  - E = edges
  - W_E = edge weights



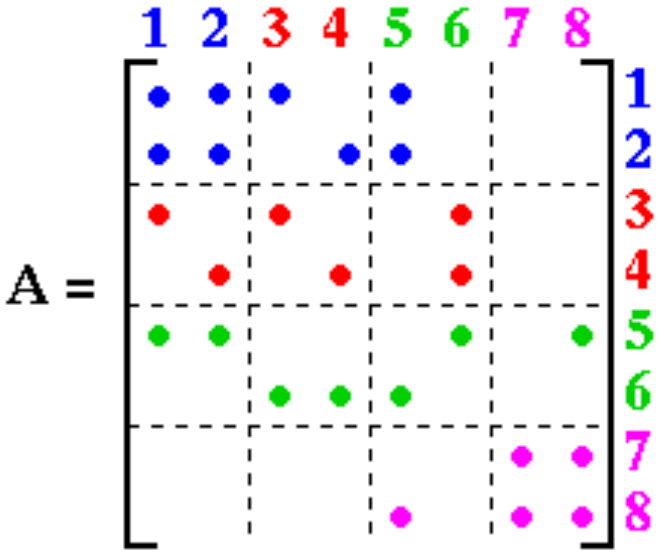
- Ex: N = {tasks}, W_N = {task costs}, edge (j,k) in E means task j sends W_E(j,k) words to task k
- Choose a partition N = N_1 U N_2 U … U N_P such that
  - The sum of the node weights in each N_j is "about the same"
  - The sum of all edge weights of edges connecting all different pairs N_j and N_k is minimized (shown in black)
- Ex: balance the work load, while minimizing communication
- Special case of N = N_1 U N_2: Graph Bisection

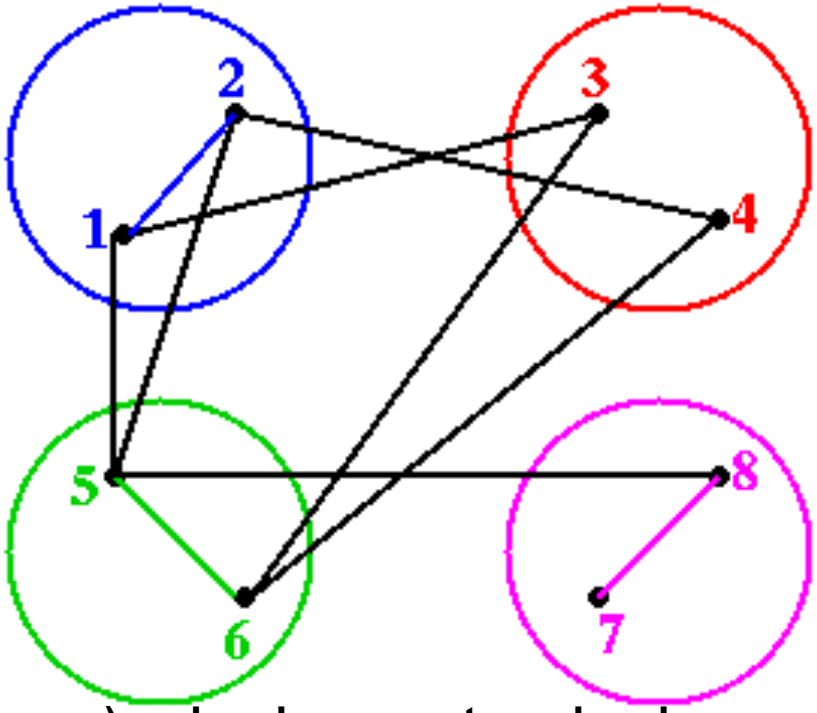
Some Applications

- Telephone network design
  - Original application, algorithm due to Kernighan
- Load Balancing while Minimizing Communication
- Sparse Matrix times Vector Multiplication (SpMV)
  - Solving PDEs
  - N = {1,…,n}, (j,k) in E if A(j,k) nonzero,
  - $W_N(j)$ = #nonzeros in row j, $W_E(j,k) = 1$
- VLSI Layout
  - N = {units on chip}, E = {wires}, $W_E(j,k)$ = wire length
- Sparse Gaussian Elimination
  - Used to reorder rows and columns to increase parallelism, and to decrease "fill-in"
- Data mining and clustering
- Physical Mapping of DNA
- Image Segmentation

Sparse Matrix Vector Multiplication y = y +A*x

Partitioning a Sparse Symmetric Matrix



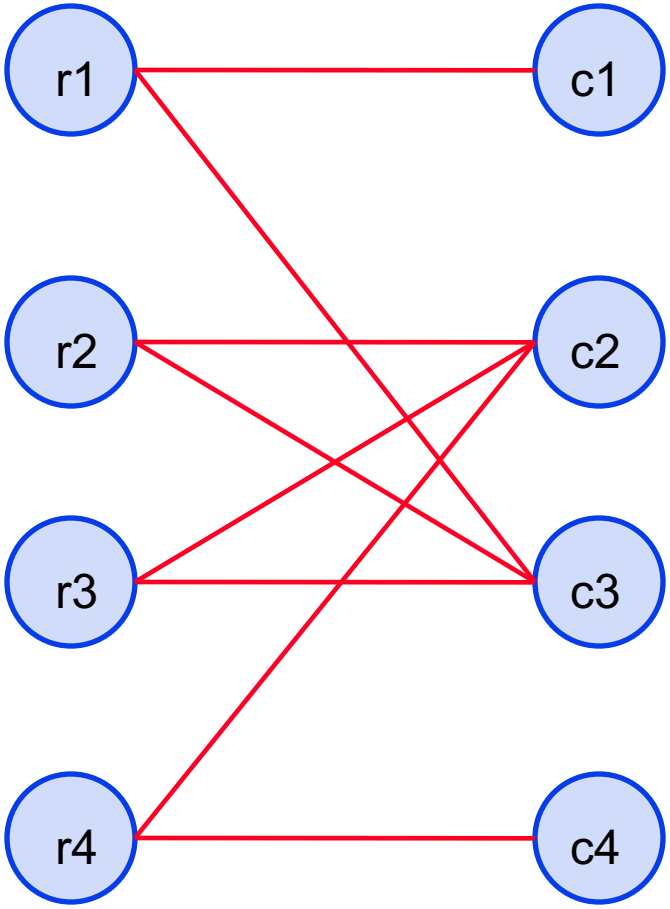



```
… declare A_local, A_remote(1:num_procs), x_local, x_remote, y_local
y_local = y_local + A_local * x_local
for all procs P that need part of x_local
        send(needed part of x_local, P)
for all procs P owning needed part of x_remote
        receive(x_remote, P)
        y_local = y_local + A_remote(P)*x_remote
```

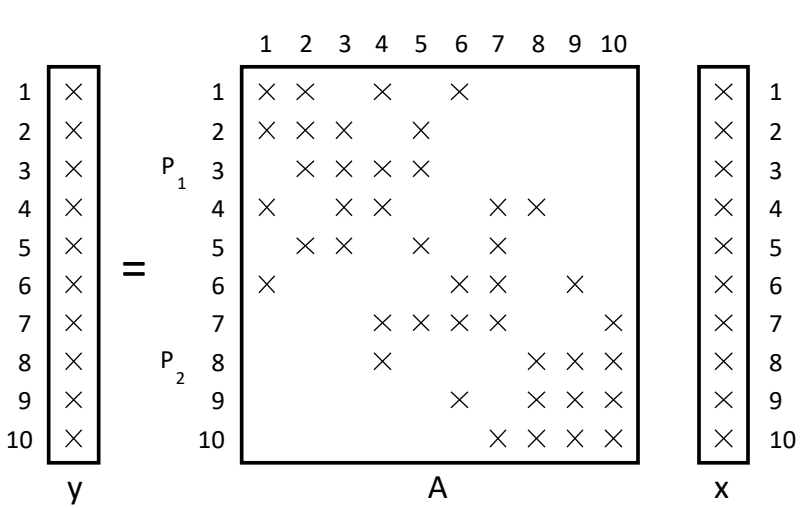
Beyond simple graph partitioning:

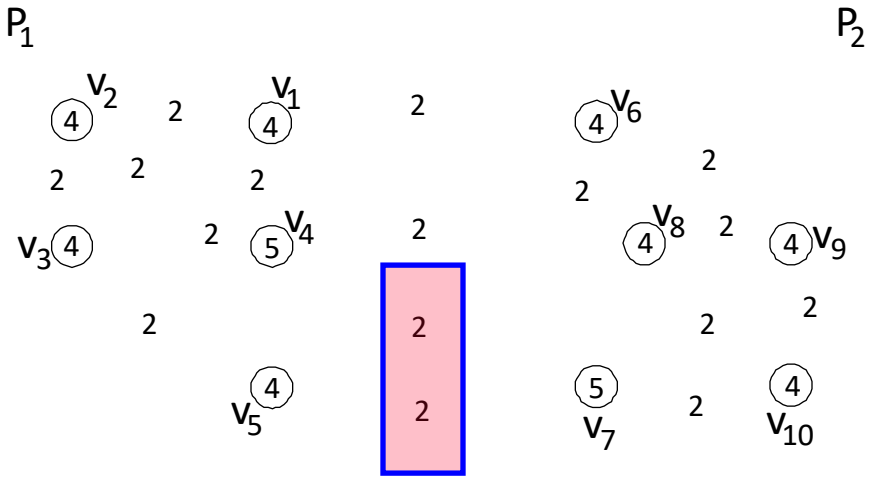
Representing a sparse matrix as a hypergraph

$$\begin{bmatrix} \times & 0 & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & 0 & \times \end{bmatrix}$$



A sparse matrix in the graph model





edge $(v_i, v_j) \in E \Rightarrow$

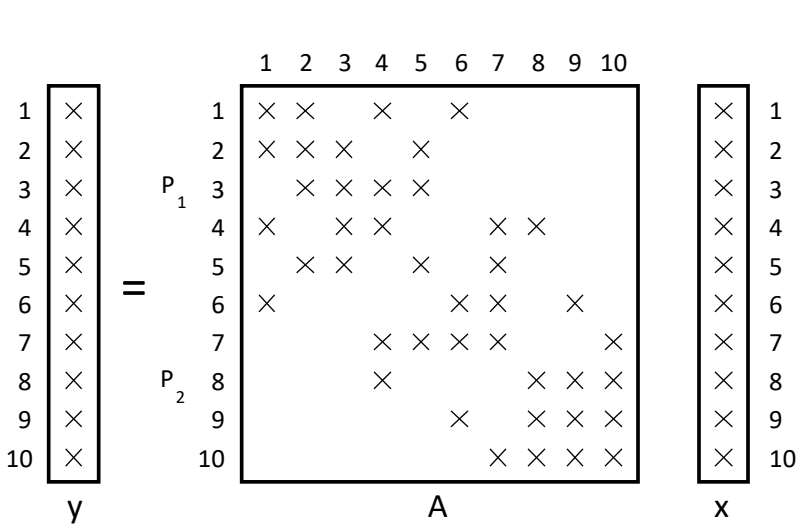
$y(i) \leftarrow y(i) + A(i,j)\, x(j)$ and $y(j) \leftarrow y(j) + A(j,i)\, x(i)$

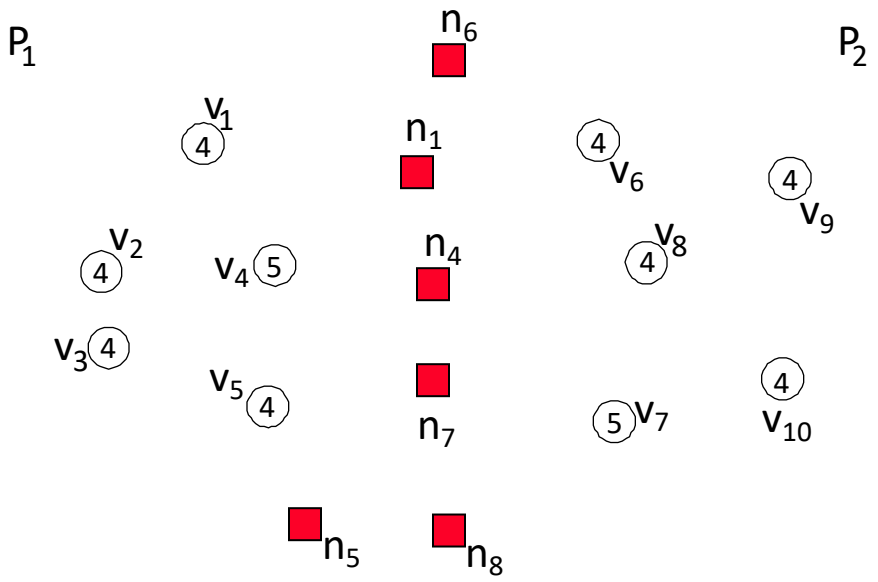
P_1 performs: $y(4) \leftarrow y(4) + A(4,7)\, x(7)$ and

$y(5) \leftarrow y(5) + A(5,7)\, x(7)$

x(7) only needs to be communicated **once !**

A sparse matrix in the hypergraph model





- Column-net model for block-row distributions
- Rows are vertices, columns are nets (hyperedges)

Each {vertex, net} pair represents unique nonzero net-cut metric:

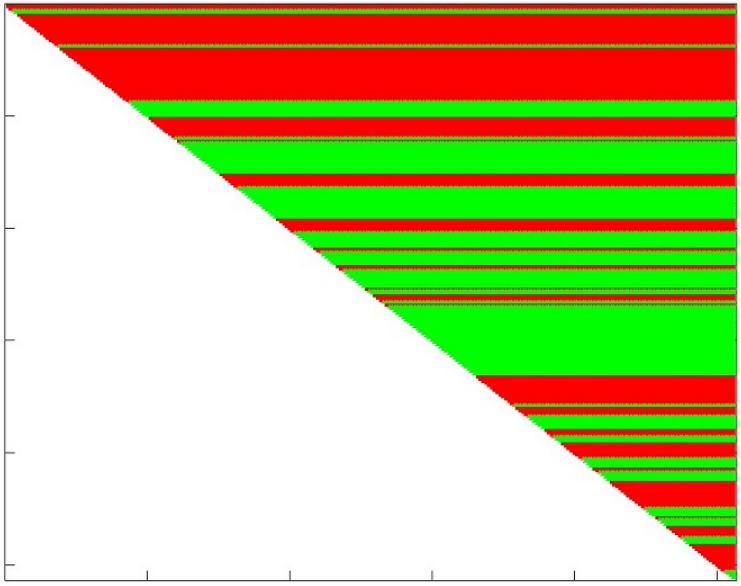
$cutsize(\Pi) = \Sigma_{n \in NE} w(n_i)$

connectivity-1 metric: $cutsize(\Pi) = \Sigma_{n \in NE} w(n_i) (c(n_j) - 1)$

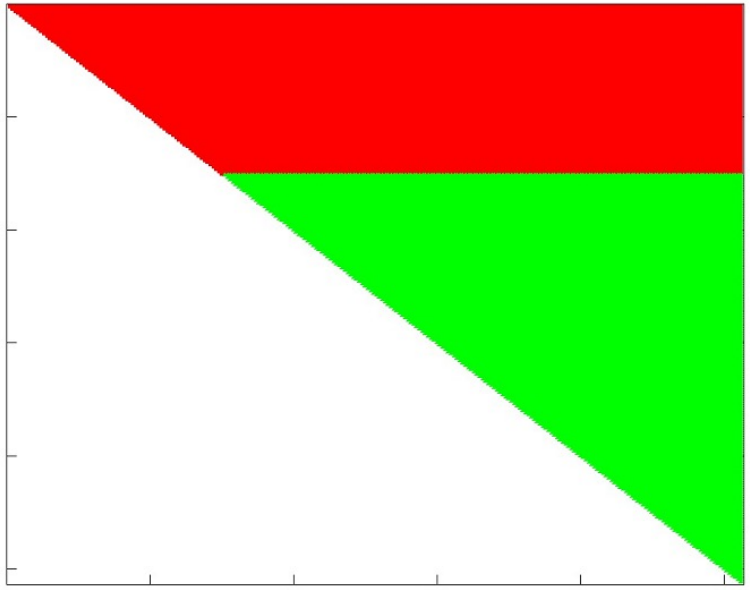
⇒ Connectivity is about the number of *parts* the hyperedge connects, not its number of pins (which is size)

Further Benefits of Hypergraph Model: Nonsymmetric Matrices

- Graph model of matrix has edge (i,j) if either A(i,j) or A(j,i) nonzero
- Same graph for A as $|A| + |A^T|$
- Ok for symmetric matrices, what about nonsymmetric?
  - Try A upper triangular



Graph Partitioning (Metis)
Total Communication Volume= 254
Load imbalance ratio = 6%



Hypergraph Partitioning (PaToH)
Total Communication Volume= 181
Load imbalance ratio = 0.1%

Summary: Graphs versus Hypergraphs

- Pros and cons
  - When matrix is non-symmetric, the graph partitioning model (using $A+A^T$) loses information, resulting in suboptimal partitioning in terms of communication and load balance.
  - Even when matrix is symmetric, graph cut size is not an accurate measurement of communication volume
  - Hypergraph partitioning model solves both these problems
  - However, hypergraph partitioning (PaToH) can be much more expensive than graph partitioning (METIS)
- Hypergraph partitioners: PaToH, HMETIS, ZOLTAN

Is Graph Partitioning a Solved Problem?

- Myths of partitioning due to Bruce Hendrickson

1. Edge cut = communication cost
2. Simple graphs are sufficient
3. Edge cut is the right metric
4. Existing tools solve the problem
5. Key is finding the right partition
6. Graph partitioning is a solved problem

- Slides and myths based on Bruce Hendrickson's:

  "Load Balancing Myths, Fictions & Legends"

- Another myth of partitioning due to Aydin Buluc

1. **Total communication volume** determines runtime
   - Max is perhaps more relevant, depending on architecture

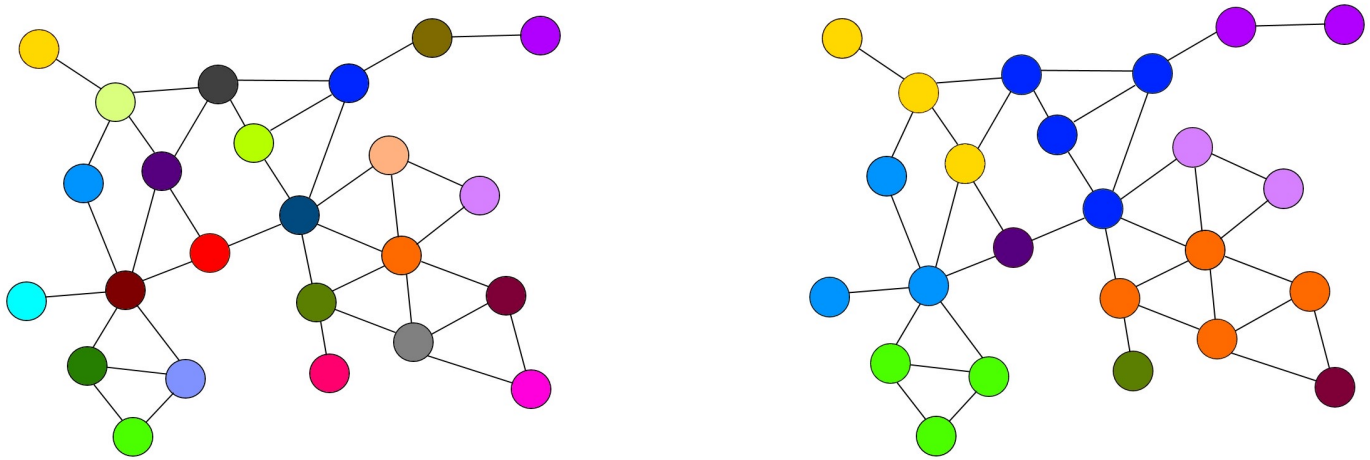
Myth: Partition Quality is Paramount

- When structure are changing dynamically during a simulation, need to partition dynamically
  - Speed may be more important than quality
  - **Partitioner must run fast in parallel**
    - Another chicken and egg problem here
  - Partition should be incremental
    - Change minimally relative to prior one
  - Must not use too much memory

- Recent research on streaming partitioning:
  - Stanton, I. and Kliot, G., “Streaming graph partitioning for large distributed graphs”. KDD, 2012.
  - The idea is used by many graph processing systems such as PowerGraph and GPS

Multilevel graph partitioning is expensive

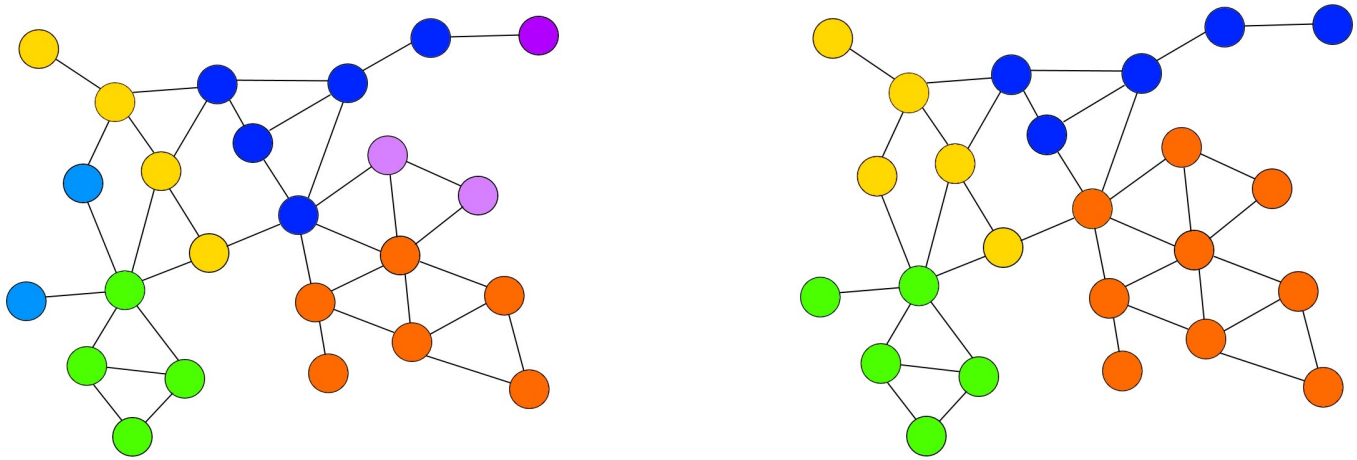
- A cheaper more scalable approach based on label propagation (example software: PULP)



- Randomly label with $n = \#verts$ labels
- Iteratively update each $v \in V(G)$ with max per-label count over neighbors with ties broken randomly

Graph Partitioning with Label Propagation

- A cheaper more scalable approach based on label propagation (example software: PULP)

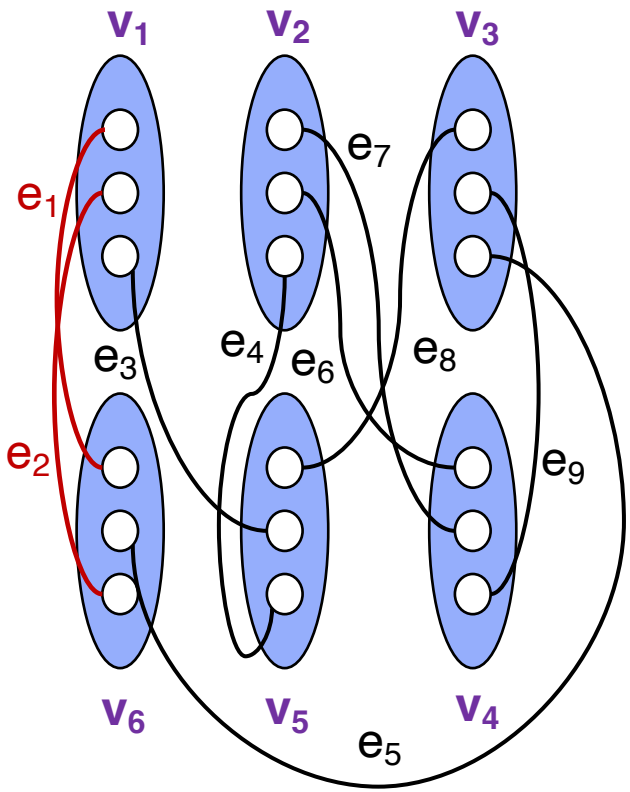


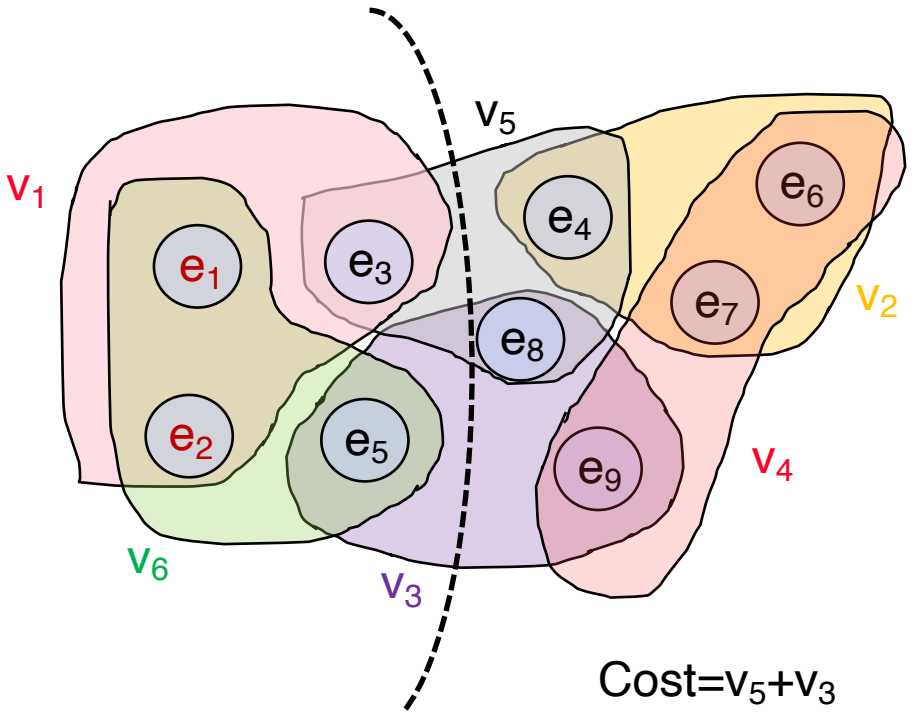
- Randomly label with $n = \#verts$ labels
- Iteratively update each $v \in V(G)$ with max per-label count over neighbors with ties broken randomly
- Algorithm completes when no new updates possible; in large graphs, fixed iteration count

Challenges of using Graph Partitioning

- You will find many surveys on Graph Partitioning
  - I helped write one too
  - Almost all content will be devoted to internals of GP algorithms
  - Multilevel, geometric, spectral, label propagation, etc.
- End user's challenge with GP is **modeling**
  - It is almost like using numerical optimization software
  - Most of the time, the user doesn't care about the internal algorithm.
  - Yes, sometimes algorithm choice matters for performance but that is secondary after one gets a version up and running
  - Almost all problems need significant modeling investment
- What is the granularity of each task?
  - These are your vertices
- What are the dependencies between tasks?
  - These are your edges
- What are you minimizing? –often the communication
- What are your constraints? –often has to do with balance

Graph/tensor contractions

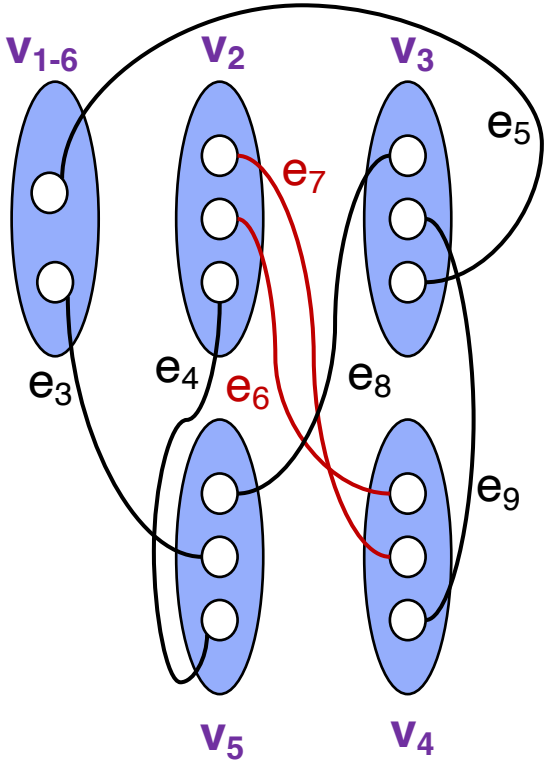


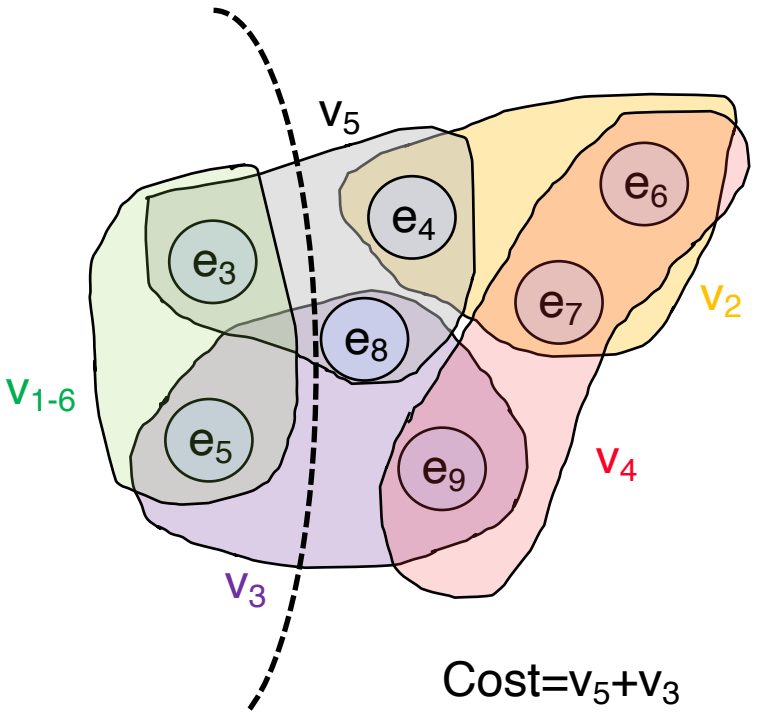


- Original graph
  - Representing tensor contractions
- Red edges will be contracted first
  - Two-index Baryon-Baryon contraction
  - $O(N^4)$ flops, $O(N^3)$ memory

- Dual hyper-graph
  - "A" potential model for GP
- Consider the bi-partitioning shown
  - **Initial contraction is "free"**
  - Because both v_1 and v_6 are entirely stored in partition A

Graph/tensor contractions

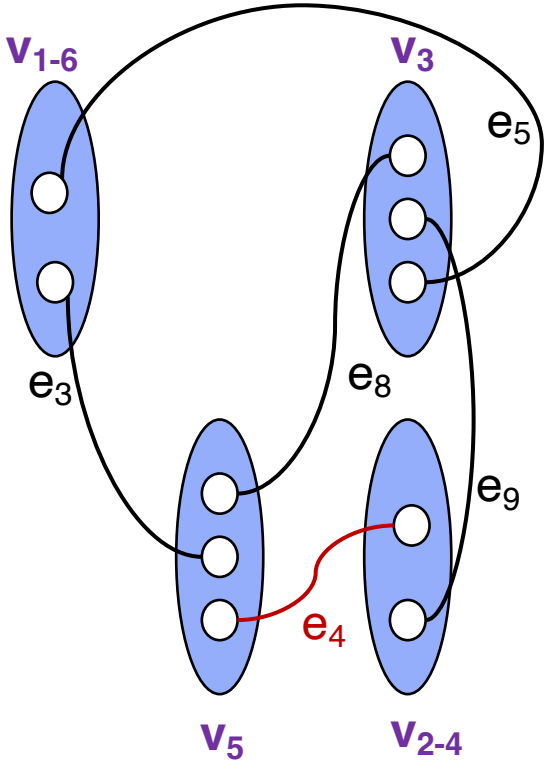


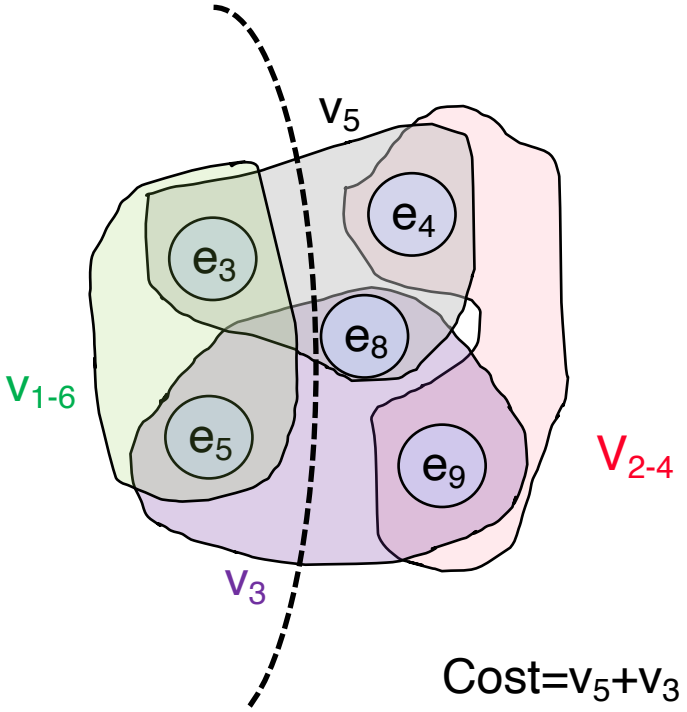


- Original graph
  - Representing tensor contractions
- Red edges will be contracted now
  - Two-index Baryon-Baryon contraction
  - $O(N^4)$ flops, $O(N^3)$ memory

- Dual hyper-graph
  - "A" potential model for GP
- Consider the bi-partitioning shown
  - **This contraction is also "free"**
  - Because both v_2 and v_4 are entirely stored in partition A

Graph/tensor contractions

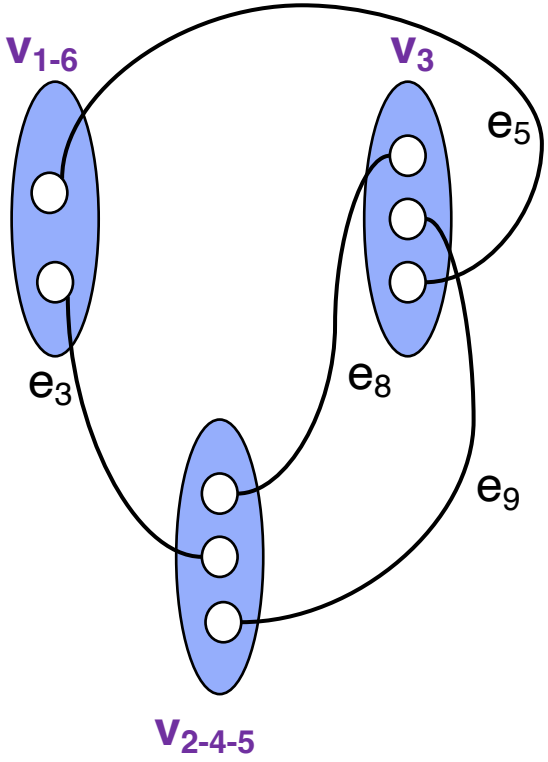


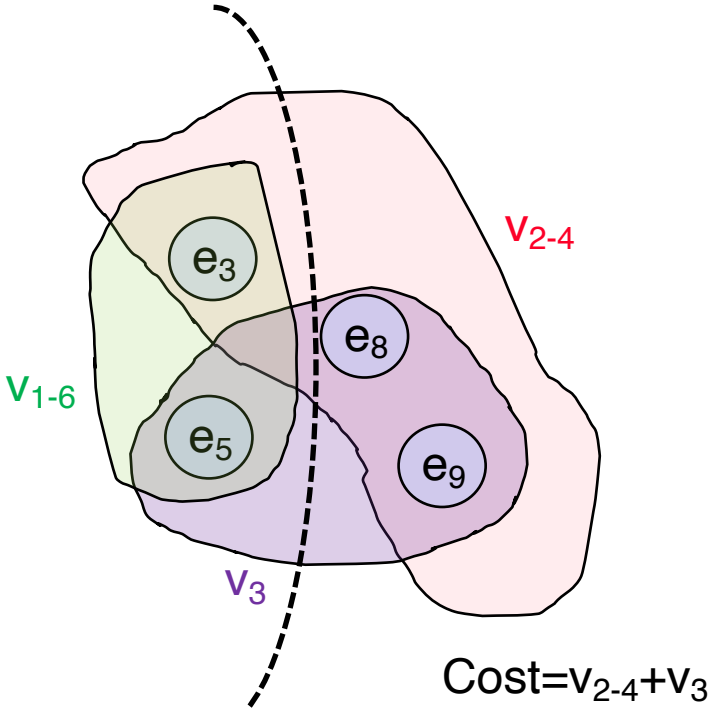


- Original graph
  - Representing tensor contractions
- Red edge will be contracted now
  - One-index Meson-Baryon contraction
  - $O(N^4)$ flops, $O(N^3)$ memory

- Dual hyper-graph
  - "A" potential model for GP
- Consider the bi-partitioning shown
  - **This contraction is NOT "free"**
  - Because both v5 spans multiple partitions; *its data needs to be replicated or communicated*

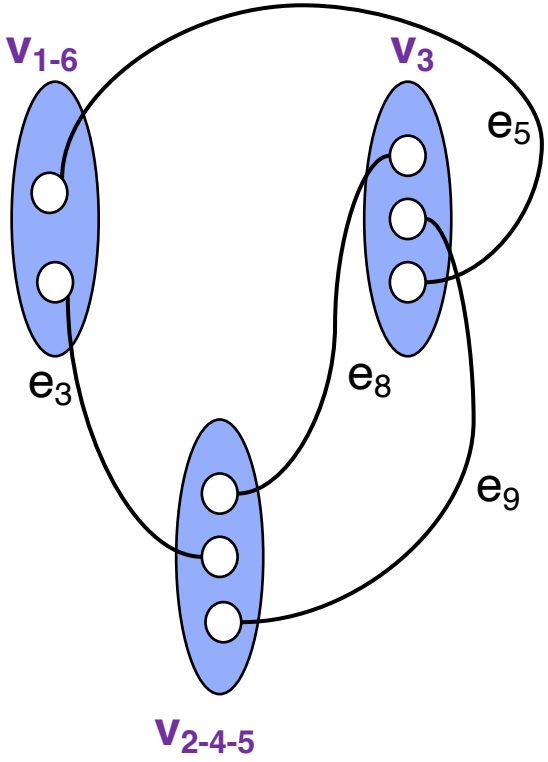
Graph/tensor contractions

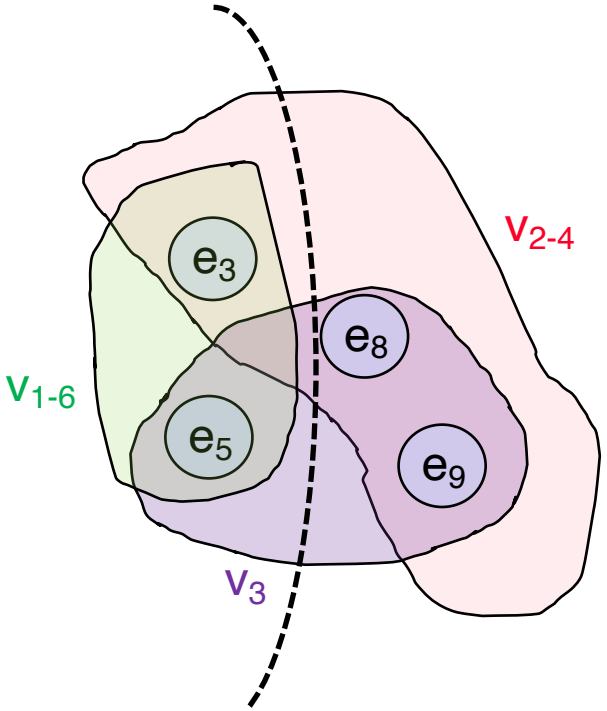




- Are we modeling this correctly? What do you think?
- What is the cost of the hyperedge v_5?
- Can we do this with graphs as opposed to hypergraphs
  - I draw it as a hypergraph solely because it avoids lots of clique-ish edges and it is easier to see the duality)

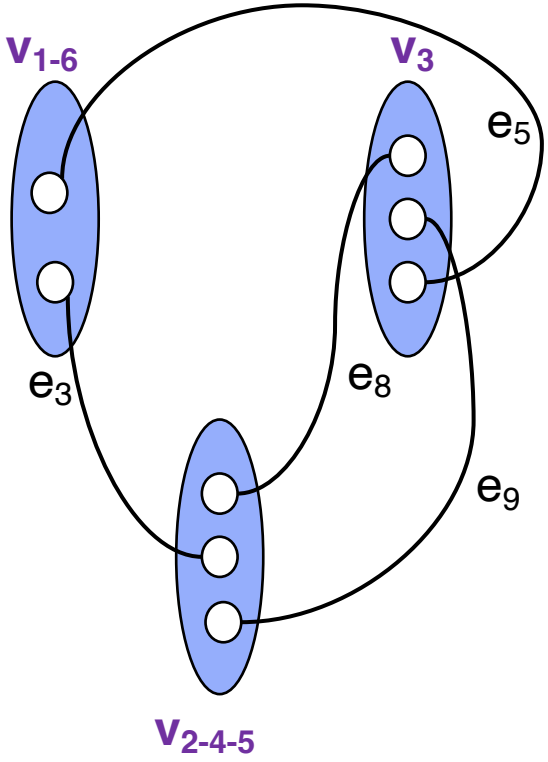
Graph/tensor contractions

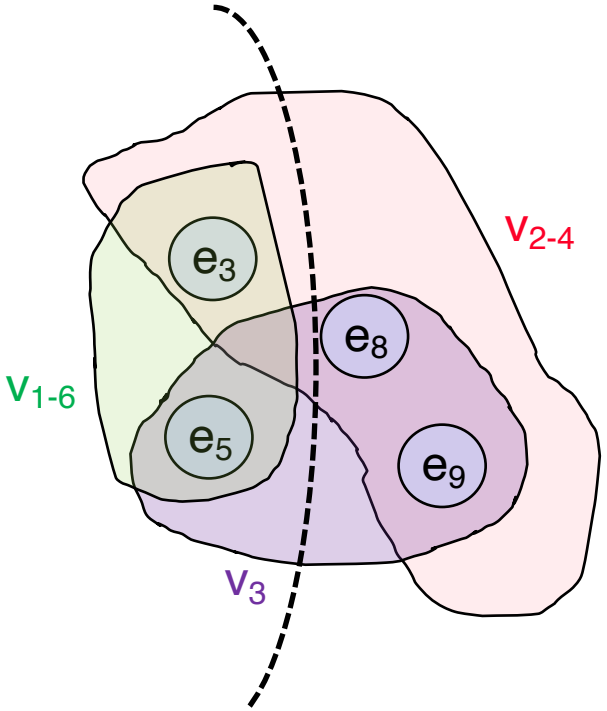




- This modeling has the following issue:
  - New hyperedges are formed at each contraction, hence our initial partitioning can not predict the future costs
  - In other words, the situation is dynamic. Similar challenge to what happens when trying to partition for graph traversal

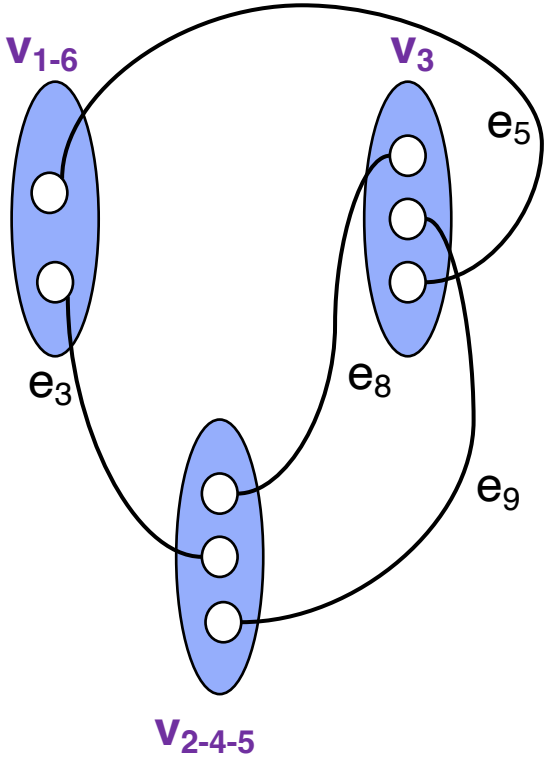
Graph/tensor contractions

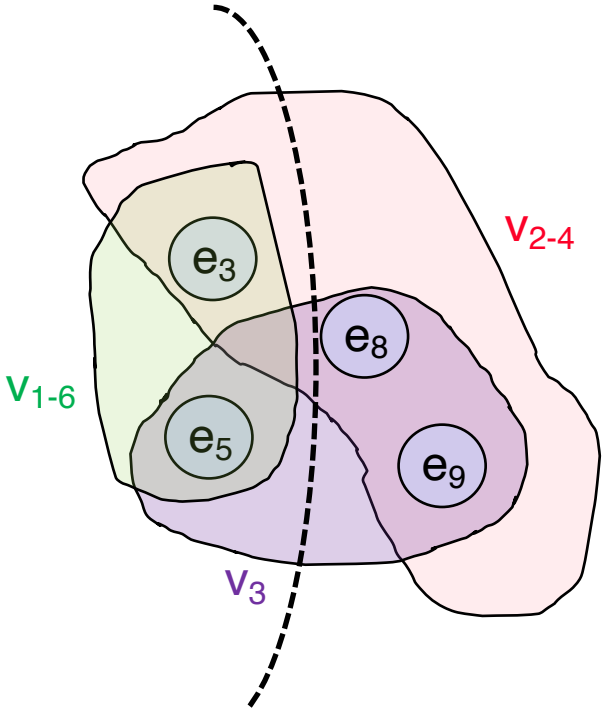




- The contractions are expensive but maybe simulating them is less so?
- Can we simulate the contraction orders and model future hyperedges accordingly?
- Likely not feasible, as there are exponentially many future scenarios
- Maybe we can do it for a single graph but not for the gigantic DAG of all graphs

Graph/tensor contractions





- Does repartitioning help? Or would it also incur the same communication?

What I didn't cover

- Multiple graphs – Jie's talk
  - Change weights
  - Much larger problem
- Determinant ordering
  - Need to sync with Will
- Finding isomorphic graphs
  - I had an intern this summer working on a distributed memory version
  - We need some representative inputs from Jlab to test the code
  - GPU version next step

Team

Aydin Buluc
Senior Scientist, LBNL
Adjunct Faculty, EECS, UC Berkeley
http://passion.lbl.gov

Oguz Selvitopi
Research Scientist (Career)
LBNL

Expert in graph and hypergraph partitioning methods, GPU algorithms, distributed-memory algorithms