Lecture 3: Circuit Partitioning – I

Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Recap: Empirical Analysis

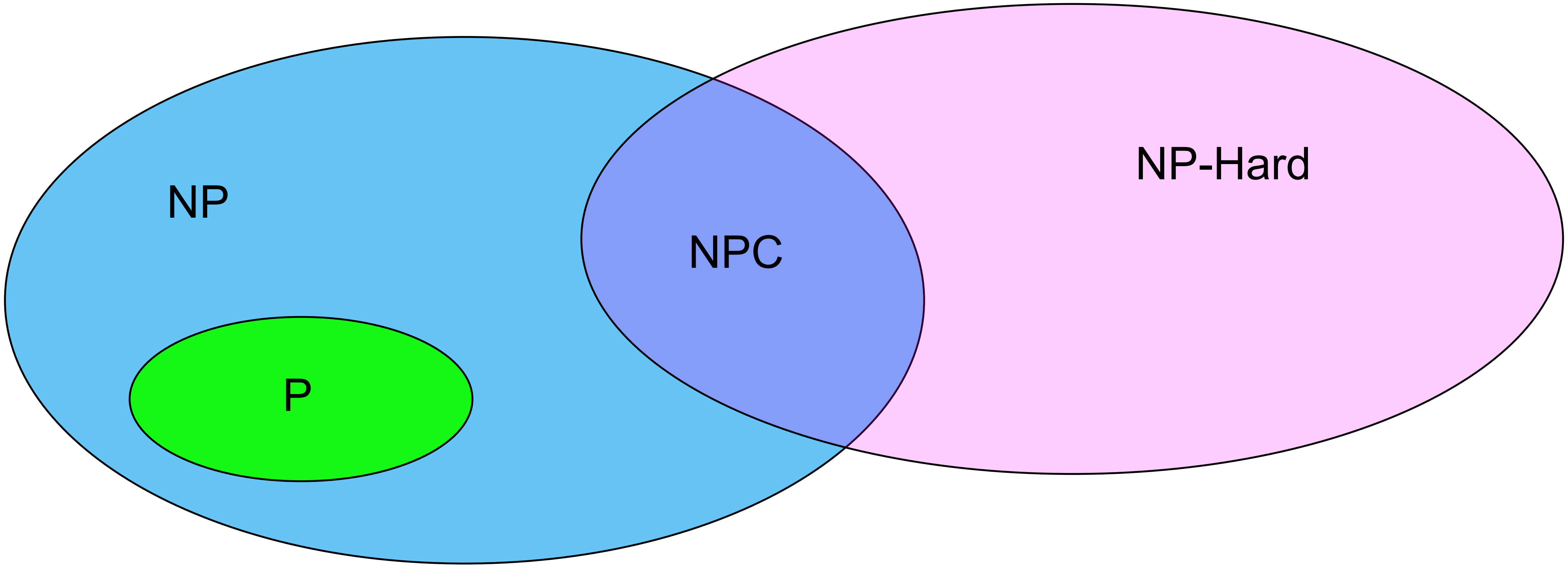
- **On modern computers (e.g., Intel i5/i7 2.5-3.5GHz)**
  - O(N) = 1000000 ~ 8000000 takes about 1s to finish
  - *Generally true* on modern PCs
- **Big-O complexity for an input size N**
  - O(N)
    - `for(int i=1; i<=N; i++) some_constant_work();`
  - $O(N^2)$

    -
      ```
      for(int i=1; i<=N; i++)
          for(int j=1; j<=N; j++)
              some_constant_time_work();
      ```
  - $O(N^3)$, $O(N^4)$…

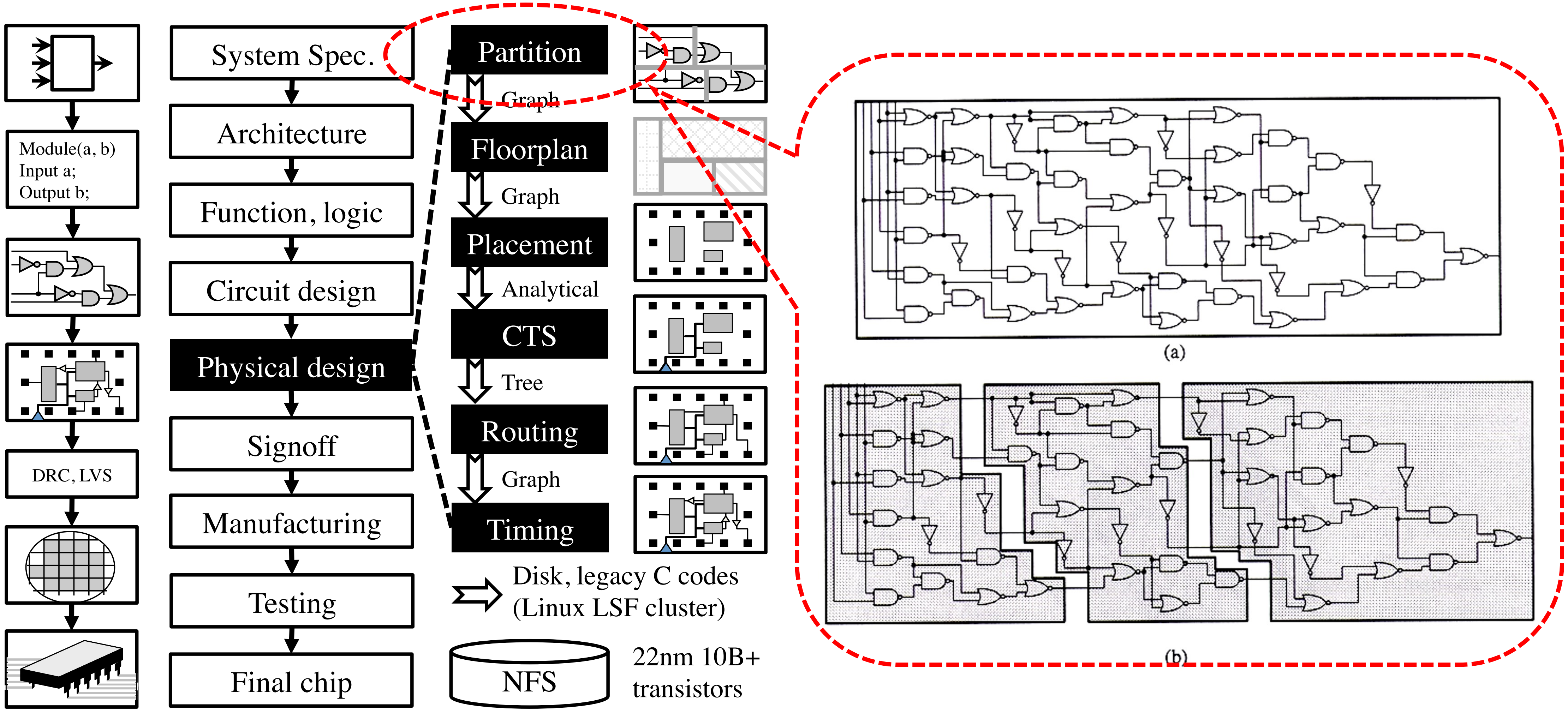
How long does these loops take?
(when N = 1000 & 1000000)

Recap: World of NP (Assuming P != NP)

There is a strong belief that $P \neq NP$, due to the existence of NPC problems.

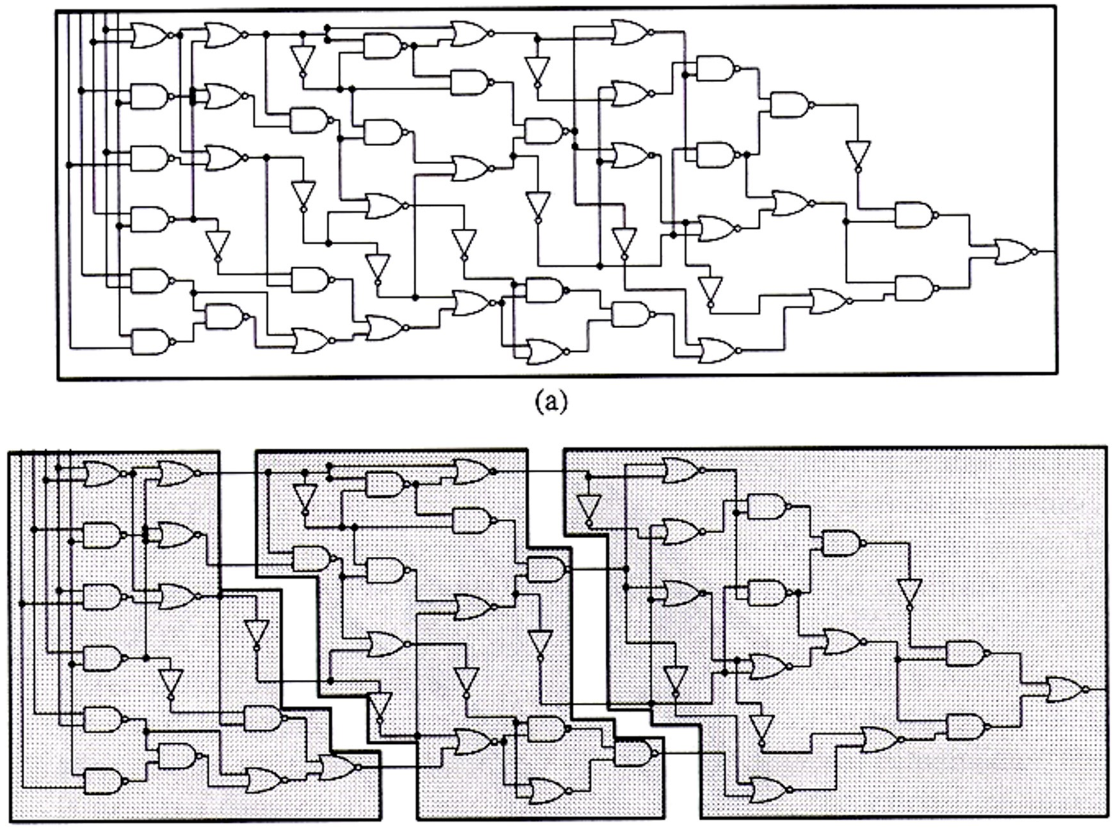


Physical Design Flow



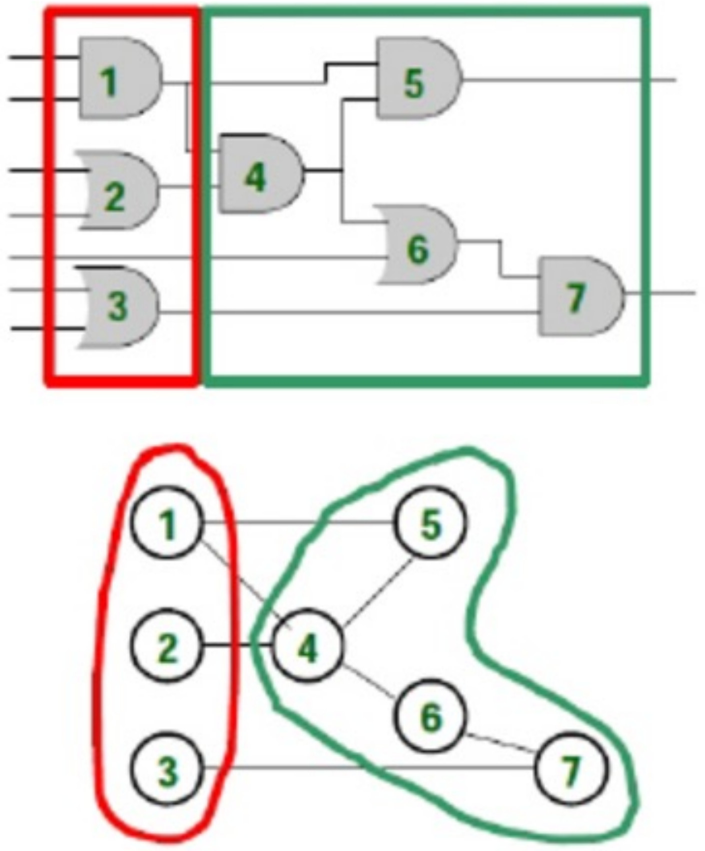
Circuit Partition

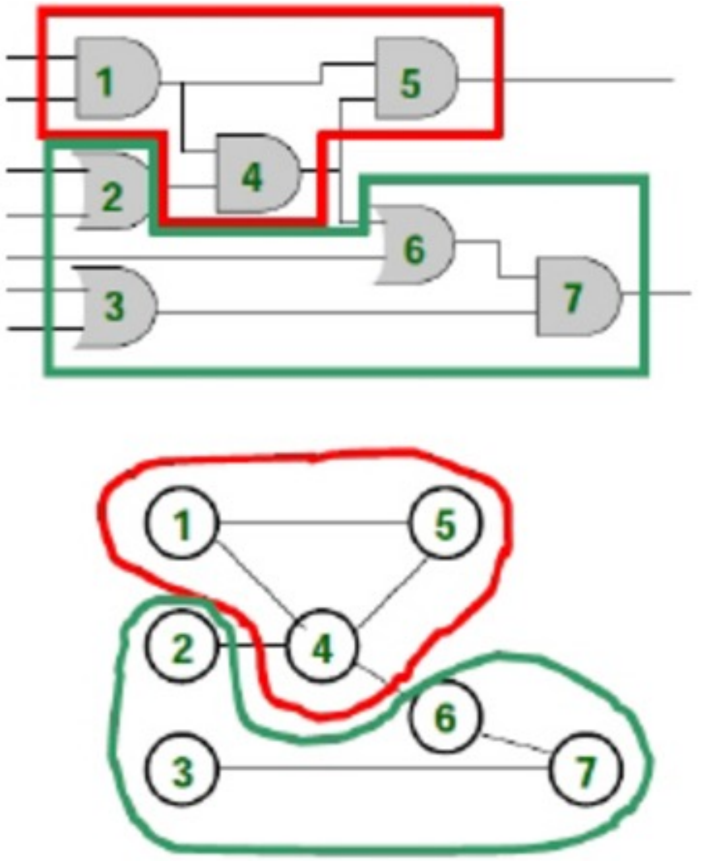
- **An essential step for reducing algorithm design complexity**
  - Divide and conquer (D&C)
- **Input**
  - A circuit graph
- **Output**
  - A set of partitioned subgraphs
- **Objective**
  - Minimize cross-connection



Circuit Partition Example

- Which partition is better in terms of cross-connection count?

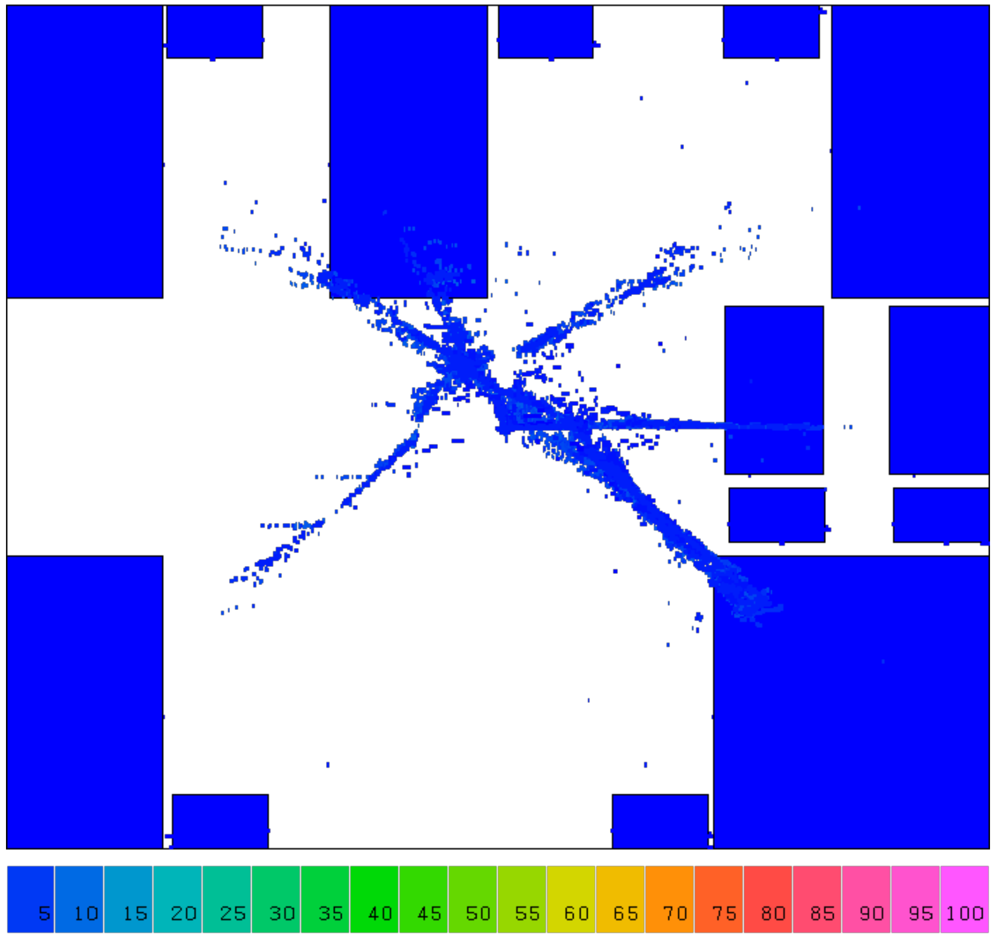




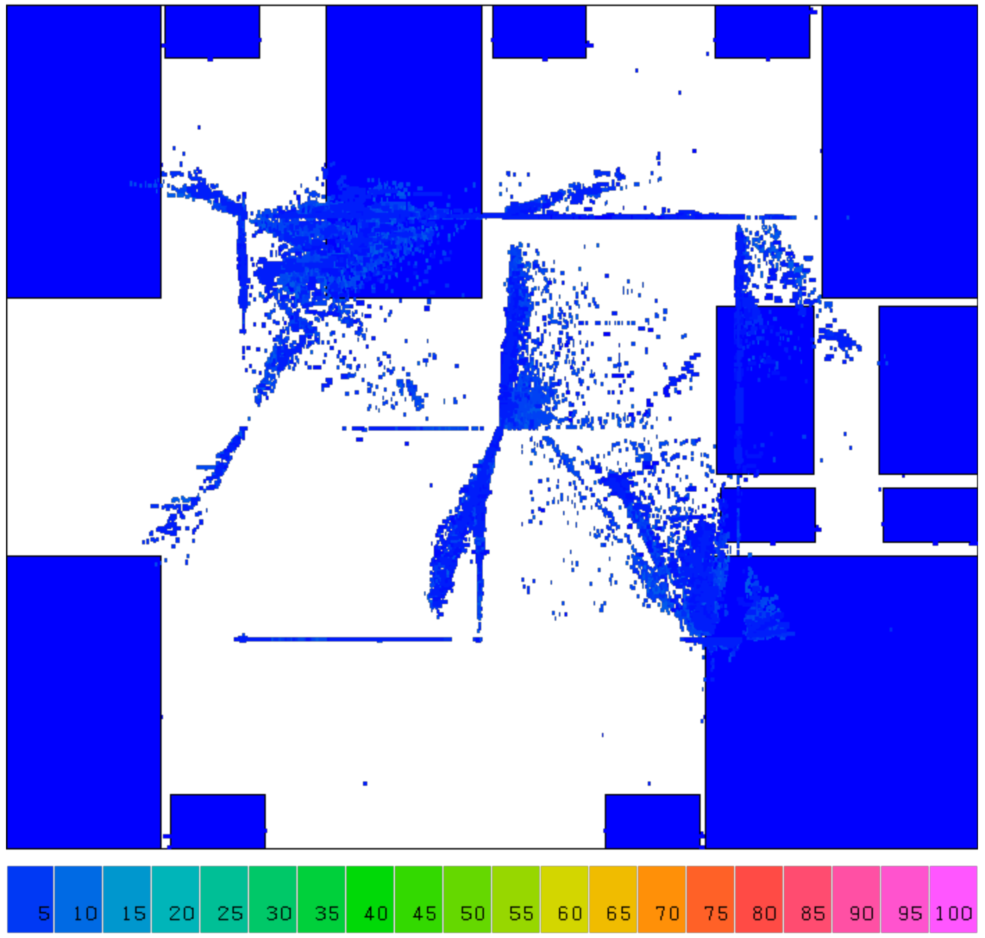
Importance of Circuit Partition

- **Divide & Conquer design methodology**
  - The most effective way to solve problems of high complexity
  - Ex: min-cut based placement, partitioning-based test generation, …
- **System-level partitioning for multi-chip designs**
  - Inter-chip interconnection delay dominates system performance
- **Circuit emulation/parallel simulation**
  - Partition large circuit into multiple FPGAs or multiple special-purpose processors
- **Parallel CAD development**
  - Task decomposition and load balancing among partitions

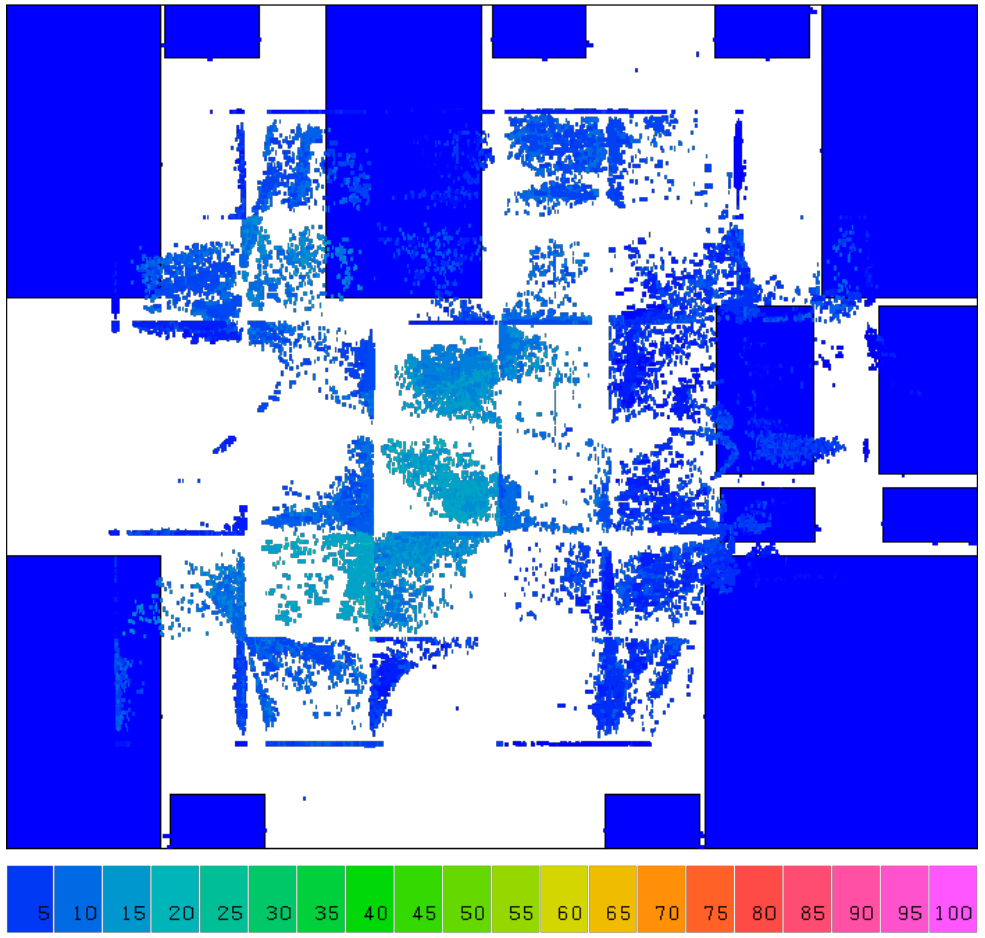
Application: D&C-based Placement



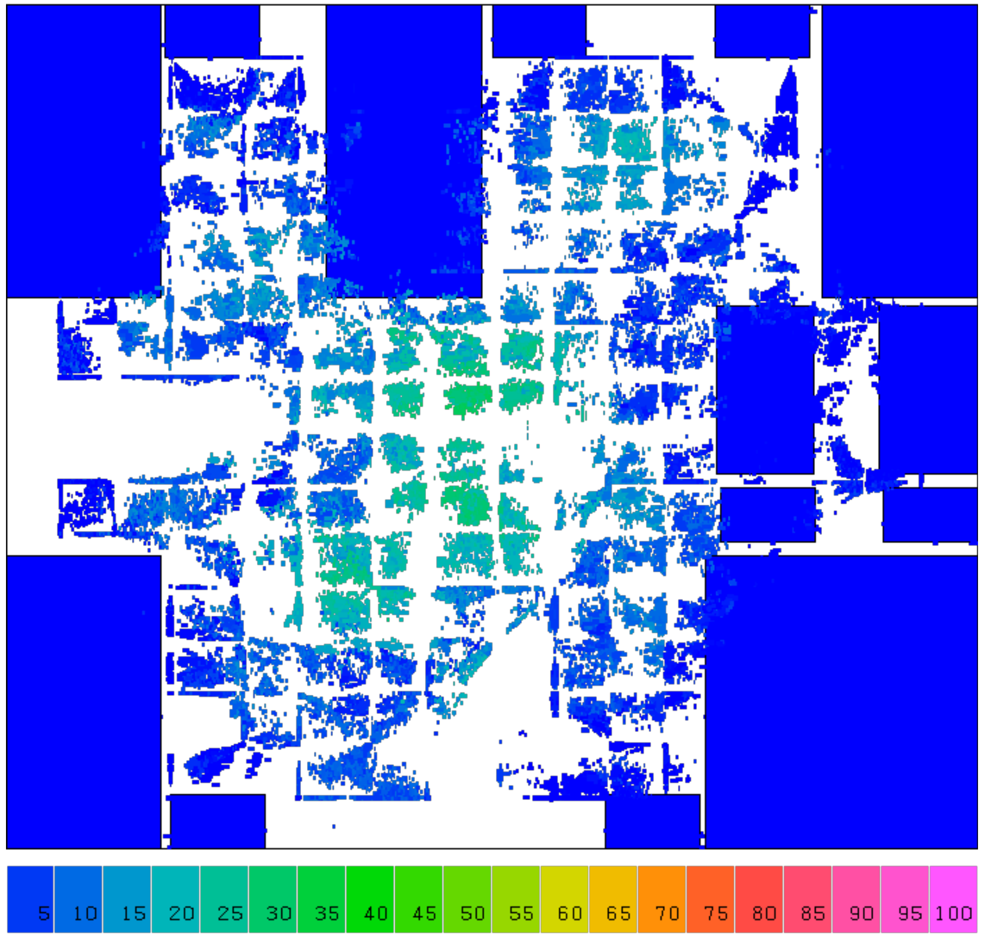
Application: D&C-based Placement



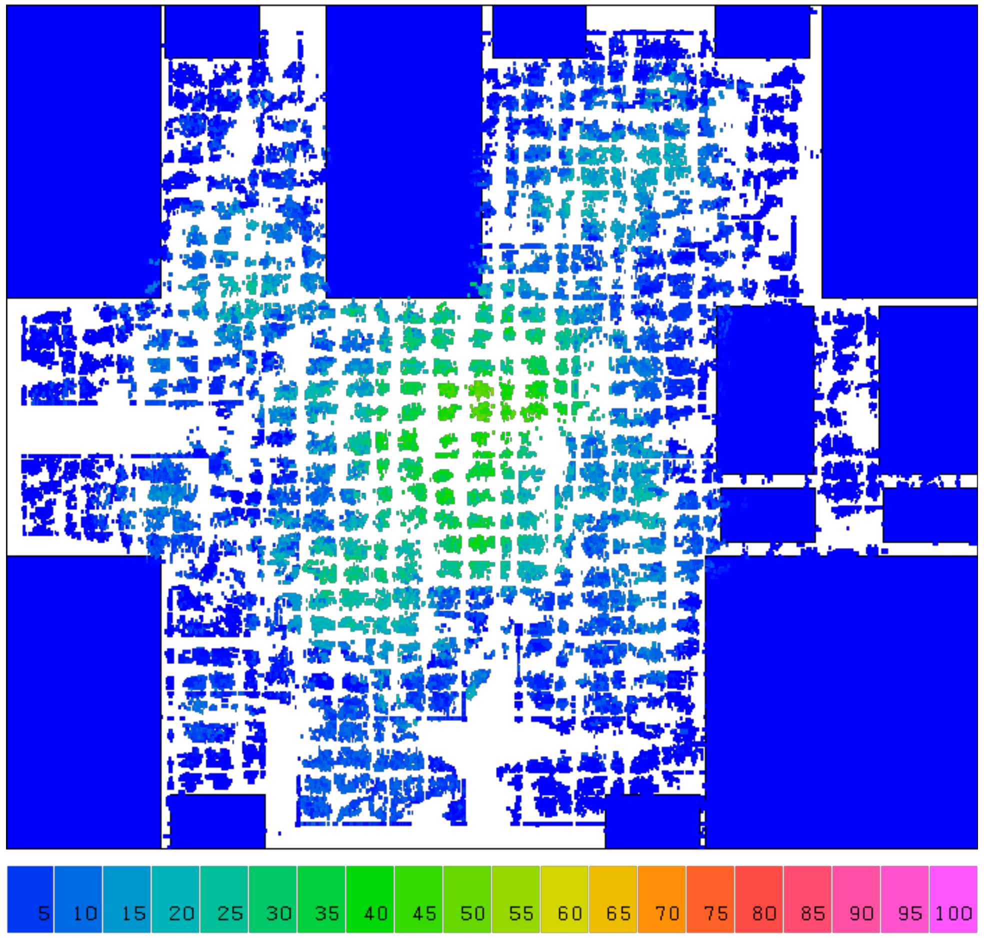
Application: D&C-based Placement



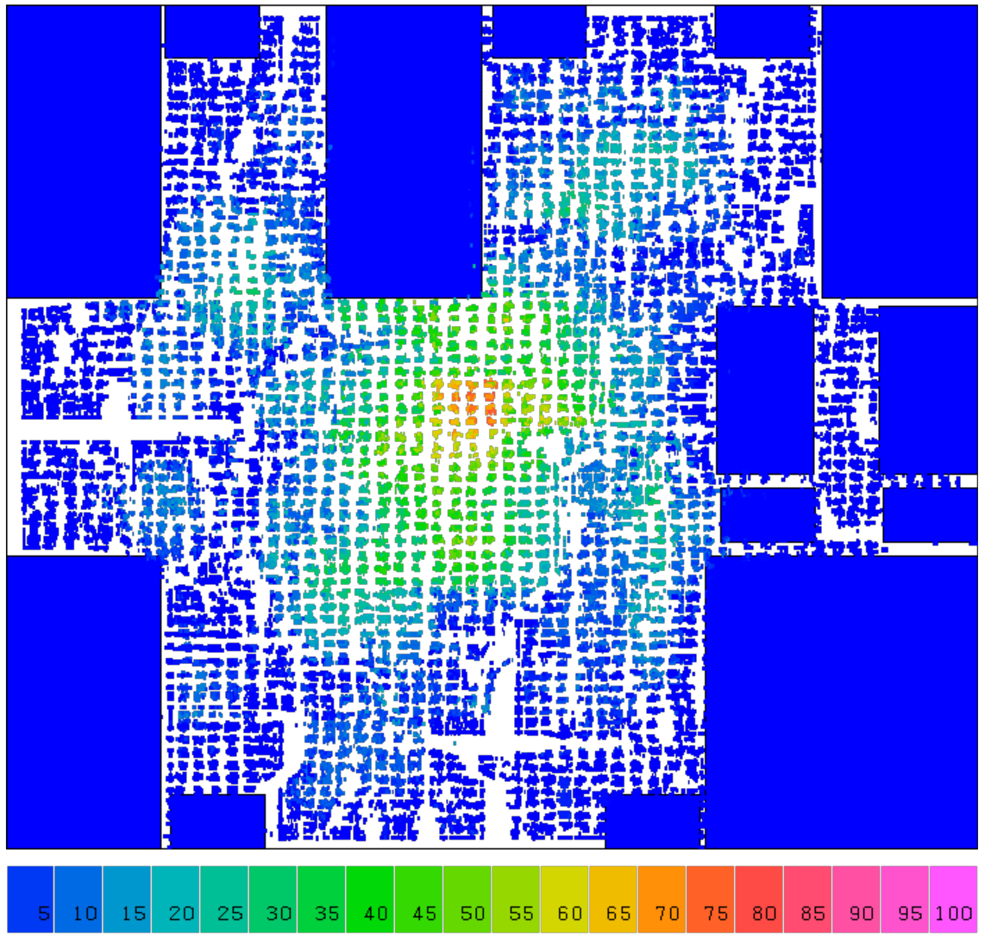
Application: D&C-based Placement



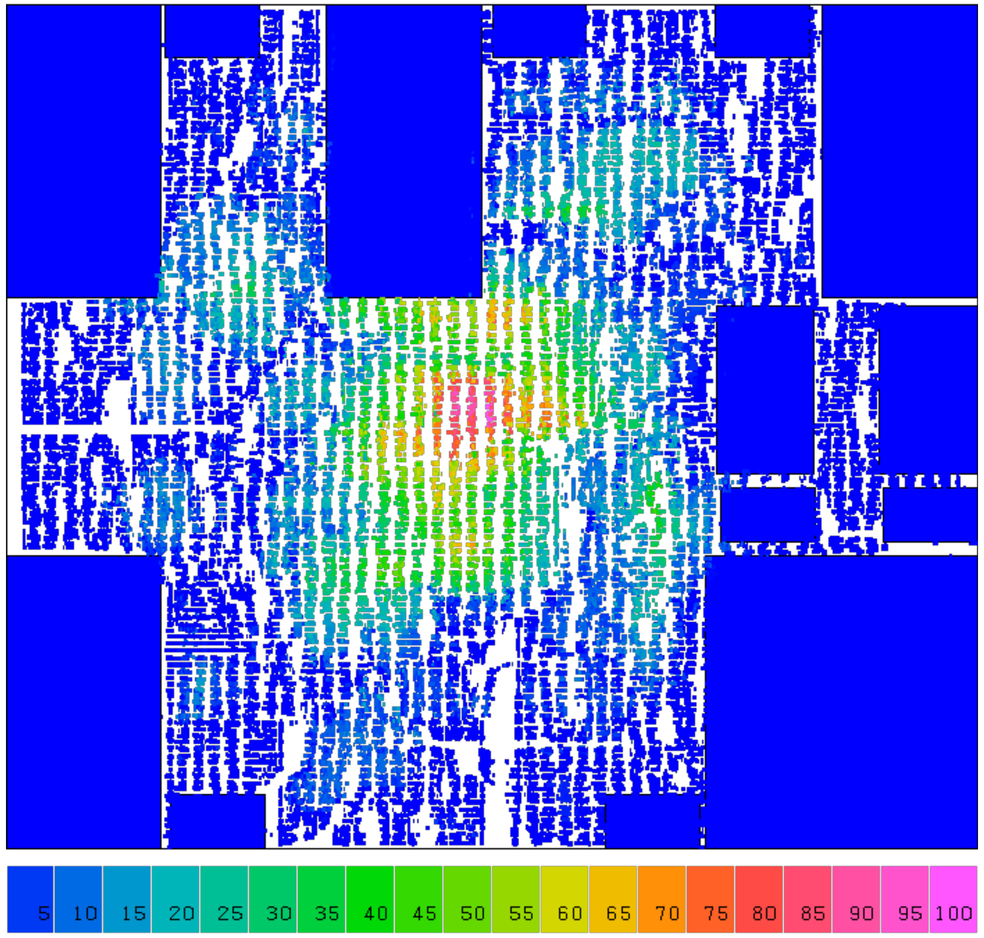
Application: D&C-based Placement



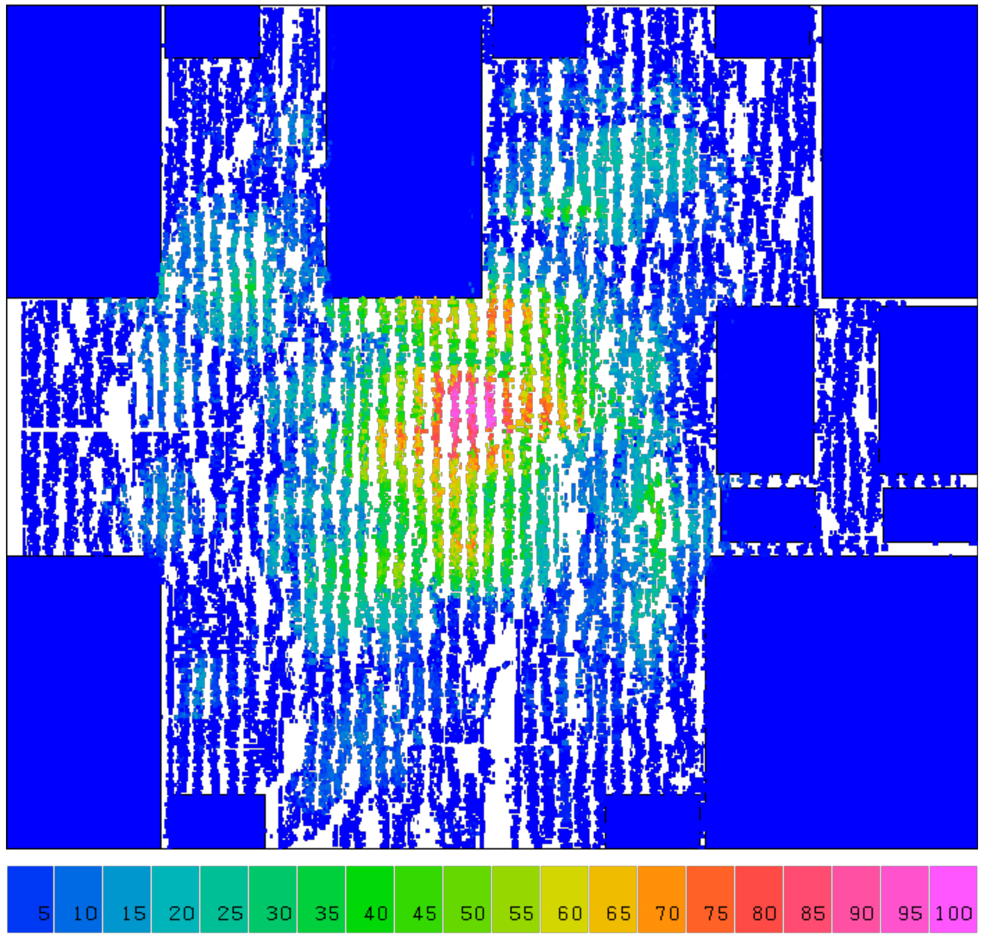
Application: D&C-based Placement



Application: D&C-based Placement



Application: D&C-based Placement



Idea Time

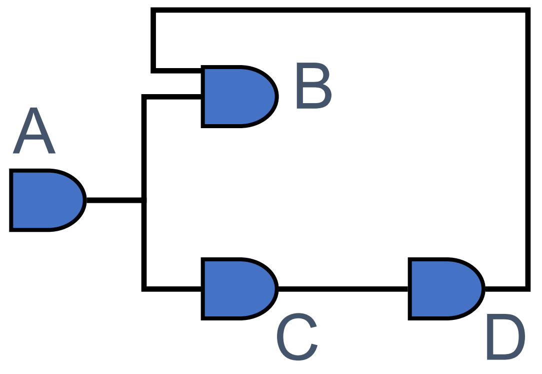
- **Input**
  - A circuit graph
- **Output**
  - Two balanced, disjoint partitions
- **Objective**
  - Minimize the cross-edge count
- **How would you solve it?**

Terminology

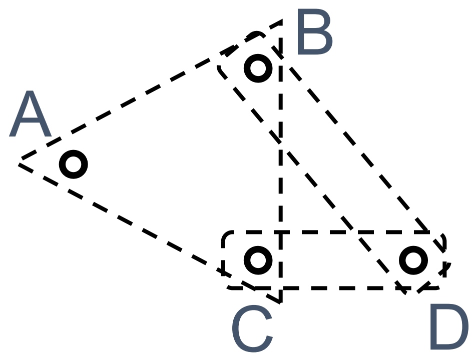
- **Partitioning**
  - Dividing bigger circuits into a small number of partitions (top down)
- **Clustering**
  - Cluster small cells into bigger clusters (bottom up)
- **Covering / Technology Mapping**
  - Clustering such that each partition (cluster) has special structure
- **k-way Partitioning**
  - Dividing into *k* partitions
- **Bipartitioning**
  - 2-way partitioning (e.g., k=2, two-way partitioning)
- **Bisectioning**
  - Bipartitioning two partitions that have the same (or balanced) size

Circuit Representation

- **Circuit netlists**
  - Gates: A, B, C, D
  - Nets: {A,B,C}, {B,D}, {C,D}

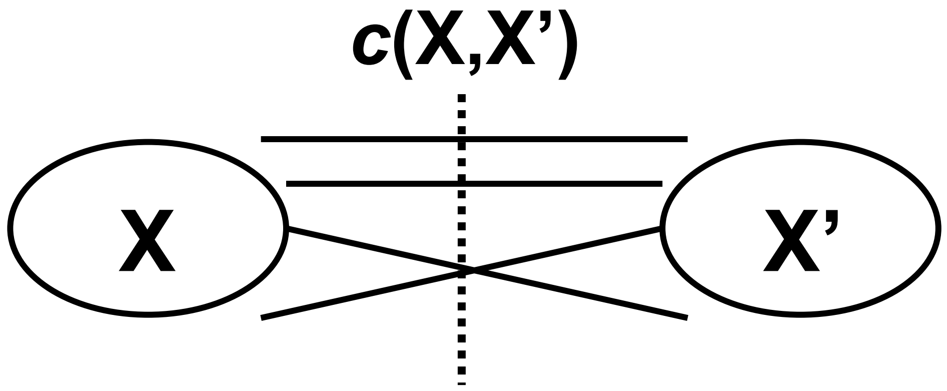


- **Hypergraph**
  - Vertices: A, B, C, D
  - Hyperedges: {A,B,C}, {B,D}, {C,D}
  - Vertex label: Gate size/area
  - Hyperedge label: Importance of net (weight)



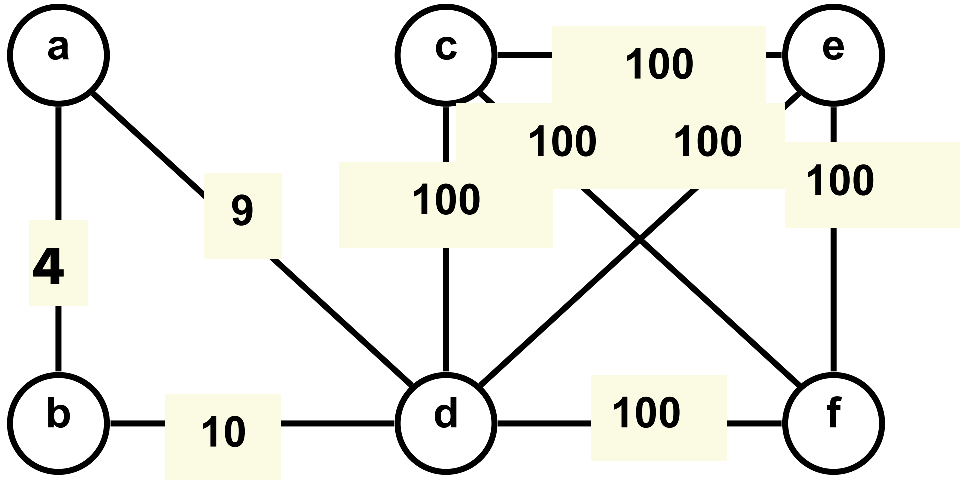
Circuit Partition Objective Formulation

Minimize interconnections between partitions



- ❑ Minimum cut: min $c(x, x')$
- ❑ Minimum bisection: min $c(x, x')$ with $|x| = |x'|$
- ❑ Minimum ratio-cut: min $c(x, x') / |x||x'|$

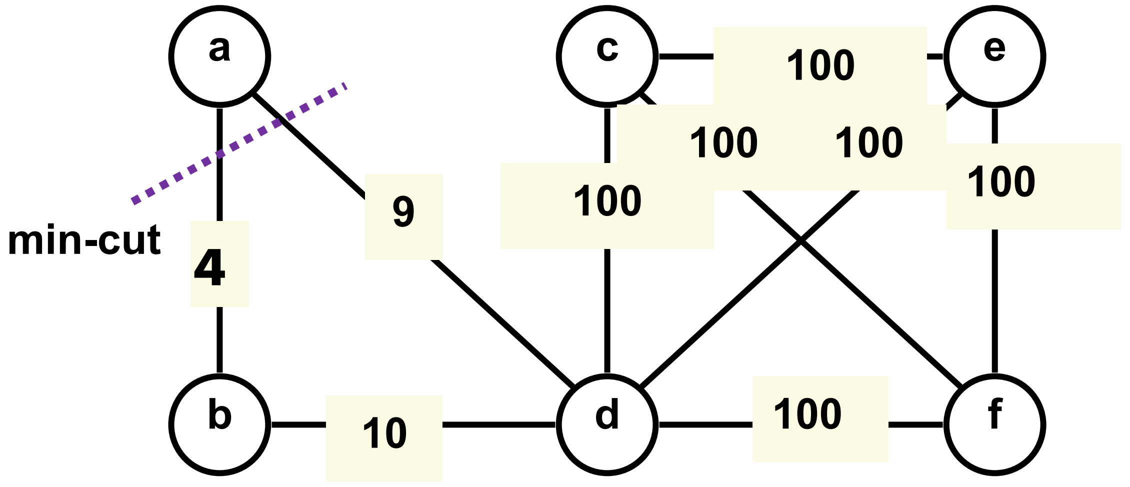
A Bi-partitioning Example



Cut cost
=
sum of cross-edge weight

Min-cut: ?
Min-bisection-cut: ?
Min-ratio-cut: ?

A Bi-partitioning Example

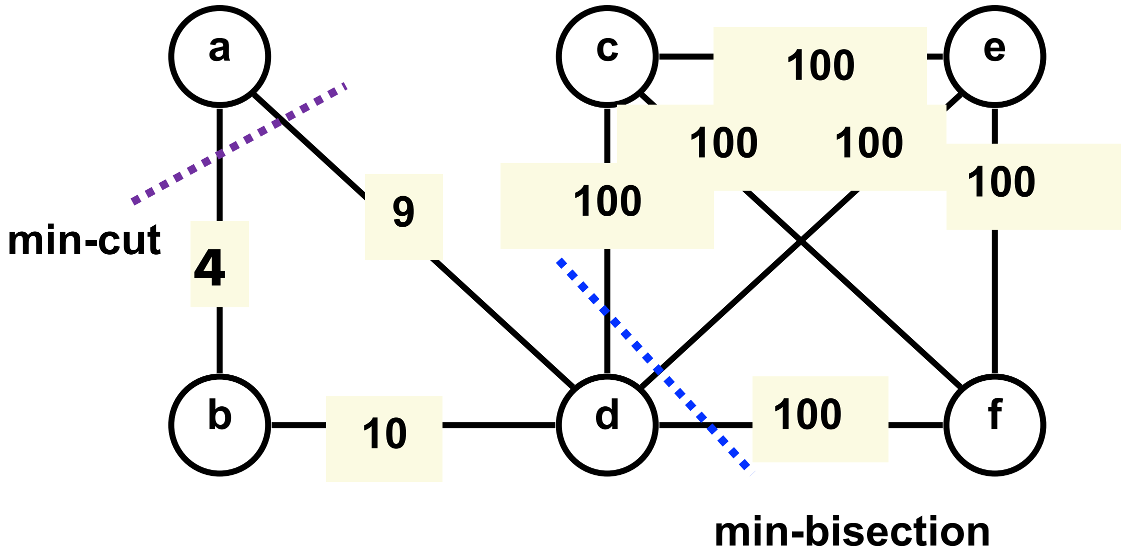


Min-cut: cut size = 13

Min-bisection-cut:

Min-ratio-cut:

A Bi-partitioning Example

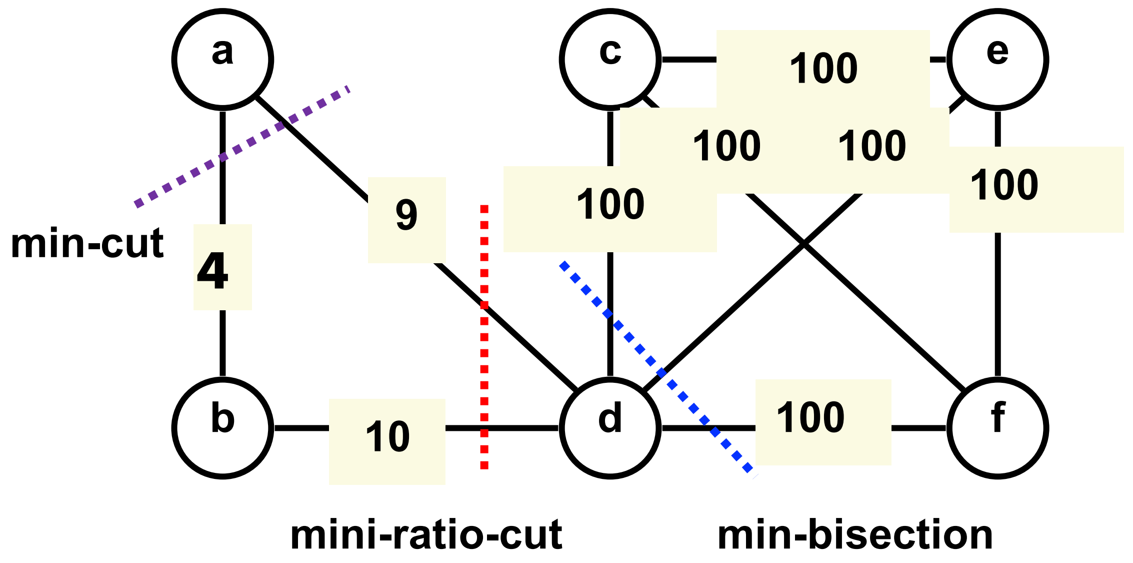


Min-cut: cut size = 13

Min-bisection-cut: cut size = 300

Min-ratio-cut:

A Bi-partitioning Example



Ratio-cut helps to identify natural clusters!

Min-cut: cut size = 13

Min-bisection-cut: cut size = 300

Min-ratio-cut: cut size = 19

Beyond Bipartition: *k*-way partition

- **General multi-way partitioning formulation**
  - Partition a network N into $N_1, N_2, \ldots, N_k$ such that
    - Each partition has an area constraint

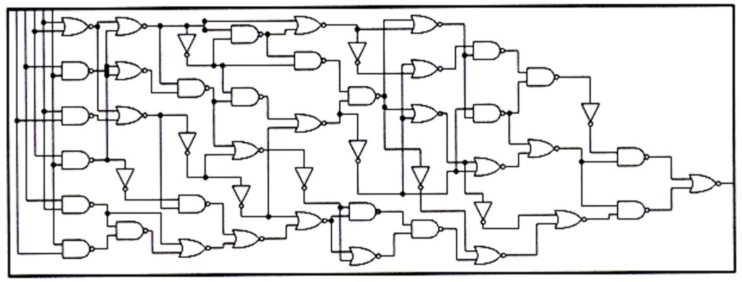
$$\sum_{v \in N_i} a(v) \leq A_i$$

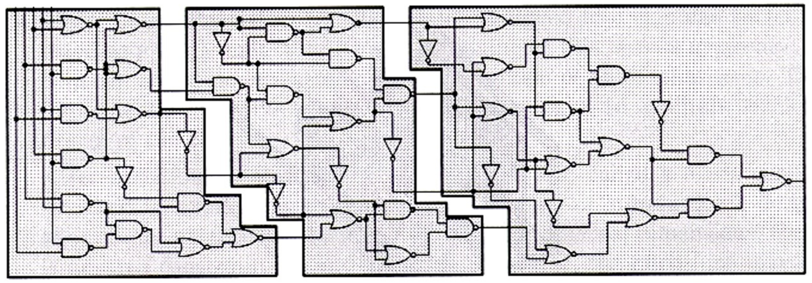
    - Each partition has an I/O constraint

$$c(N_i, N - N_i) \leq I_i$$

  - Minimize the total interconnection:

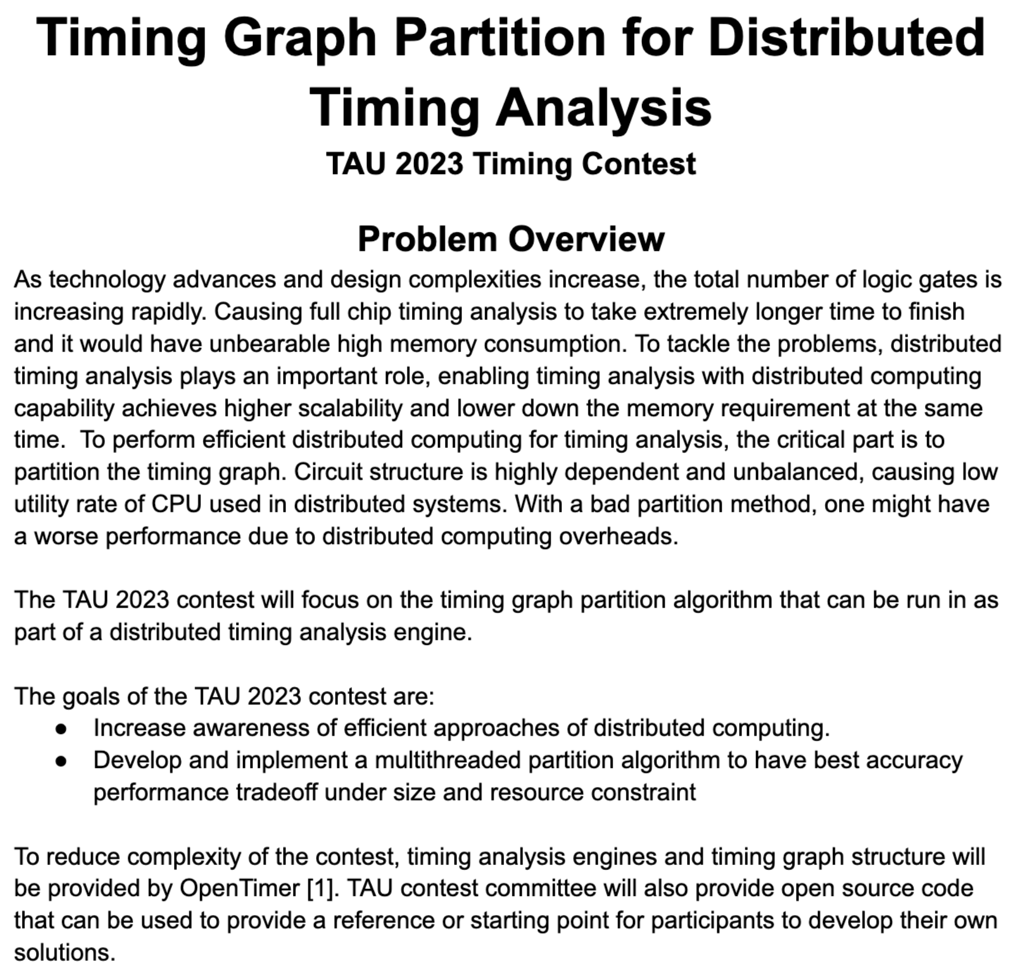
$$\sum_{N_i} c(N_i, N - N_i)$$


(a)


(b)

3-way partitions (k=3)

ACM TAU 2023 Contest: *k*-way Partition



Timing Graph Partition for Distributed Timing Analysis

TAU 2023 Timing Contest

Problem Overview

As technology advances and design complexities increase, the total number of logic gates is increasing rapidly. Causing full chip timing analysis to take extremely longer time to finish and it would have unbearable high memory consumption. To tackle the problems, distributed timing analysis plays an important role, enabling timing analysis with distributed computing capability achieves higher scalability and lower down the memory requirement at the same time. To perform efficient distributed computing for timing analysis, the critical part is to partition the timing graph. Circuit structure is highly dependent and unbalanced, causing low utility rate of CPU used in distributed systems. With a bad partition method, one might have a worse performance due to distributed computing overheads.

The TAU 2023 contest will focus on the timing graph partition algorithm that can be run in as part of a distributed timing analysis engine.

The goals of the TAU 2023 contest are:

- Increase awareness of efficient approaches of distributed computing.
- Develop and implement a multithreaded partition algorithm to have best accuracy performance tradeoff under size and resource constraint

To reduce complexity of the contest, timing analysis engines and timing graph structure will be provided by OpenTimer [1]. TAU contest committee will also provide open source code that can be used to provide a reference or starting point for participants to develop their own solutions.

Circuit Partitioning Algorithms

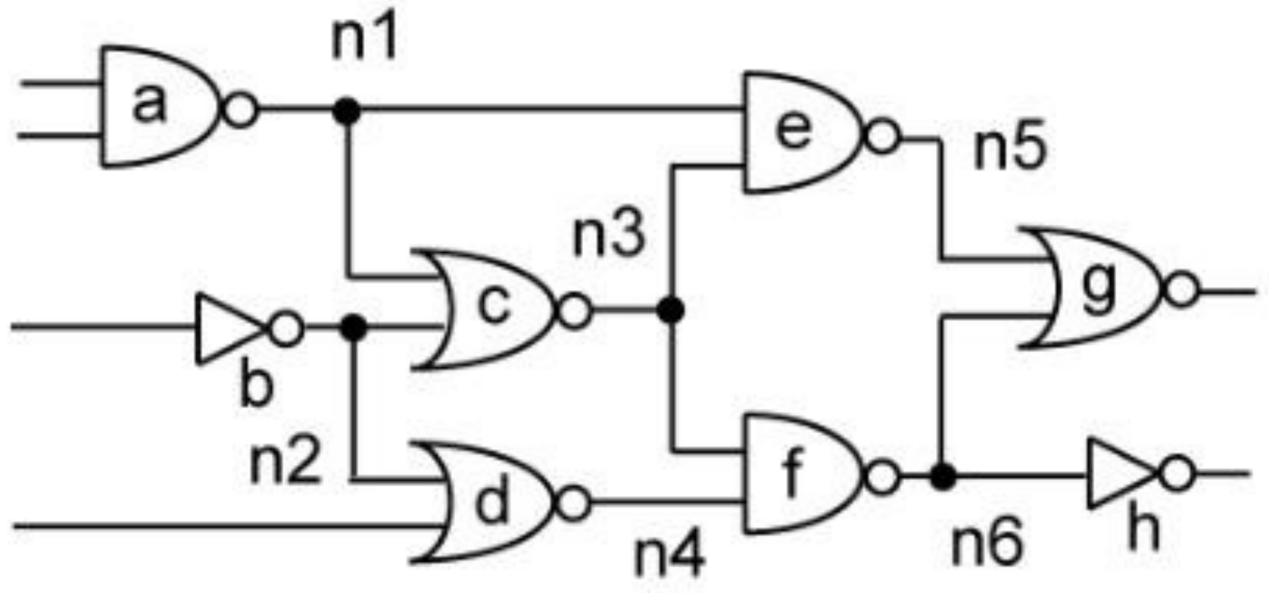
- Iterative partitioning algorithms
- Spectral based partitioning algorithms
- Deterministic vs. probabilistic algorithms
- Net partitioning vs. module partitioning
- Multi-way partitioning
- Multi-level partitioning
- Further study in partitioning techniques (timing-driven …)

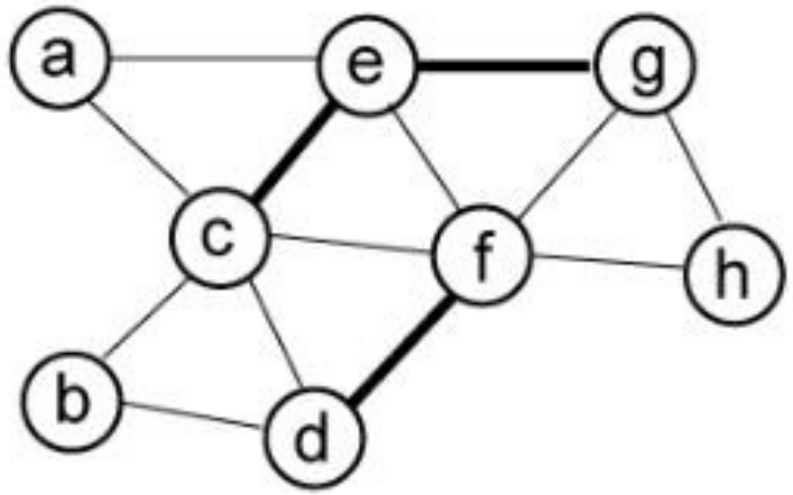
Iterative Partitioning Algorithms

- **Greedy iterative improvement method**
  - [Kernighan-Lin 1970]
  - [Fiduccia-Mattheyses 1982]
  - [Krishnamurthy 1984]
  - …
- **Simulated Annealing**
  - [Kirkpartrick-Gelatt-Vecchi 1983]
  - [Greene-Supowit 1984]
  - …

Kernighan-Lin (KL) Algorithm

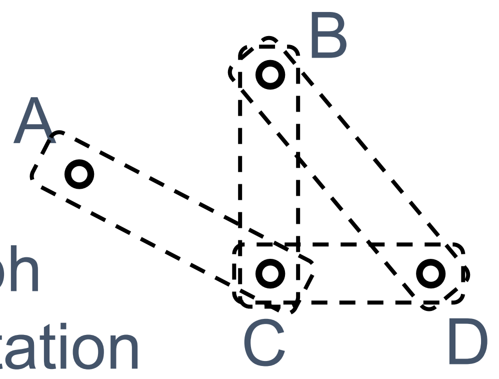
- “An Efficient Heuristic Procedure for Partitioning Graphs,” *The Bell System Technical Journal*, 49(2):291-307, 1970



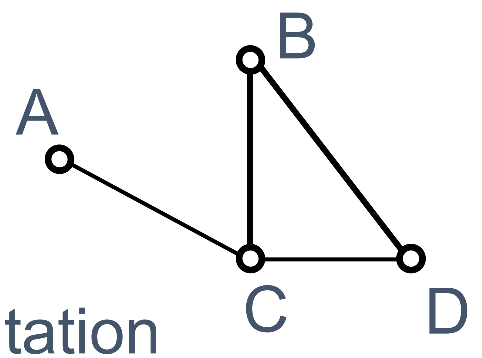


Restricted Partition Problem

- **Restrictions**
  - For Bisectioning of circuit (k=2, two-way partitioning)
  - Assume all gates are of the same size
  - Works only for 2-terminal nets
- **If all nets are 2-terminal, the hypergraph is called a graph**



Hypergraph Representation



Graph Representation

Problem Formulation

- **Input: A graph with**
  - Set vertices V ($|V| = 2n$)
  - Set of edges E ($|E| = m$)
  - Cost c_{AB} for each edge $\{A, B\}$ in E
- **Output: 2 partitions *X* & *Y* such that**
  - Total cost of edges cut is minimized
  - Each partition has n vertices
- **This problem has been proven to be NP-complete**
  - Very difficult to find the optimal partition efficiently

A Naive Approach …

- Try <u>all</u> possible bisections, find the best one
- If there are *2n* vertices,

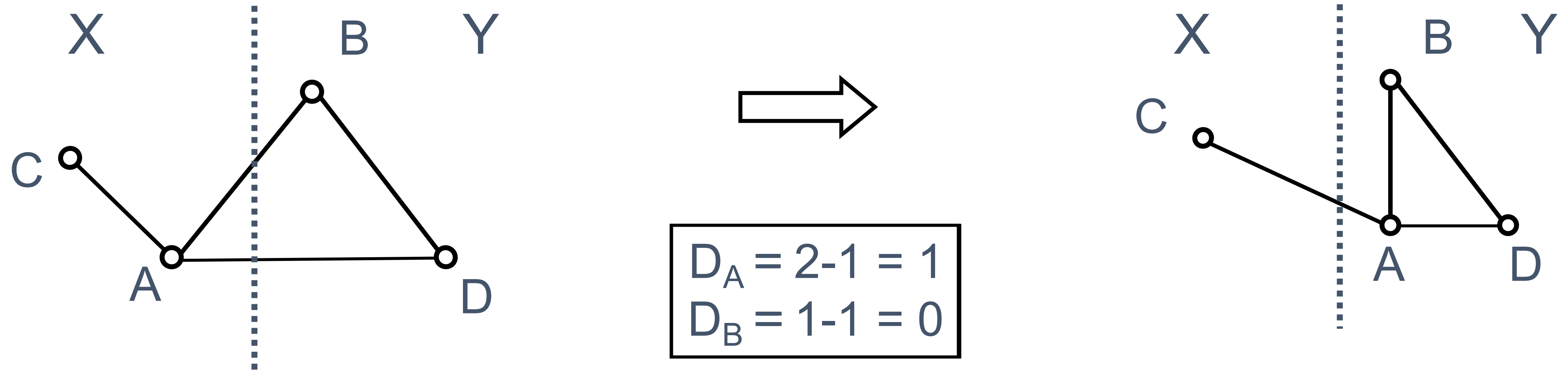
  # of possibilities = $C_n^{2n} / 2$ = $(2n)! / (2 \times (n!^2)) = n^{O(n)}$

- For 4 vertices (A,B,C,D), 3 possibilities
  1. X={A,B} & Y={C,D}
  2. X={A,C} & Y={B,D}
  3. X={A,D} & Y={B,C}

- For 100 vertices, $5\text{x}10^{28}$ possibilities
  - Need $1.59\text{x}10^{13}$ years if one can try 100M possibilities per second

Idea of KL Algorithm

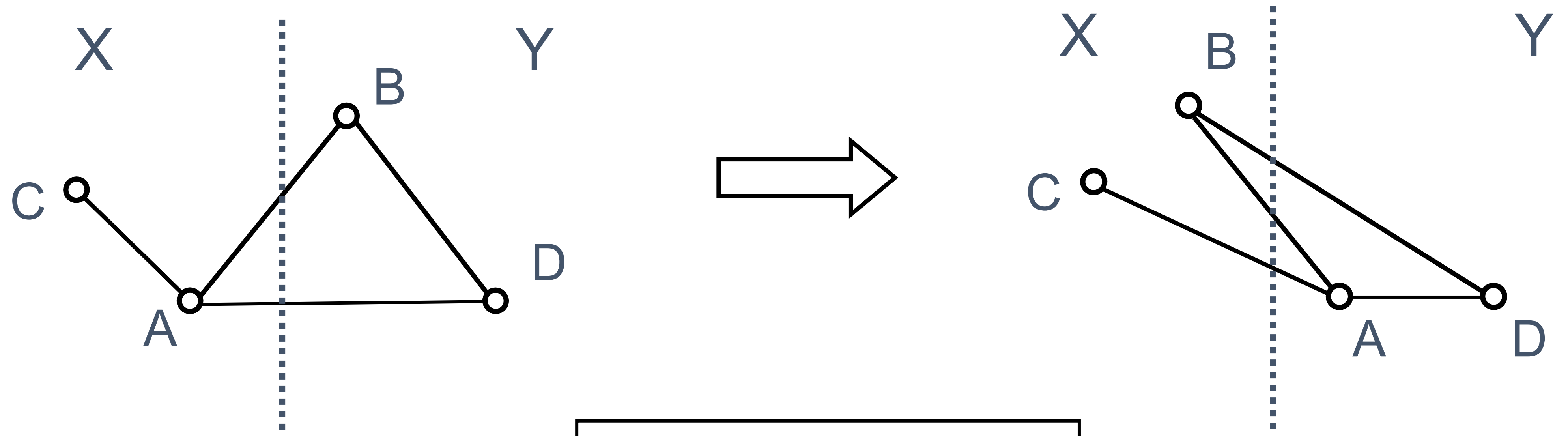
- D_A = Decrease in cut value if moving A
  - External cost (connection) E_A – Internal cost I_A
  - Moving node a from block X to block Y would increase the value of the cutset by E_A and decrease it by I_A



D_A = 2-1 = 1
D_B = 1-1 = 0

Idea of KL Algorithm

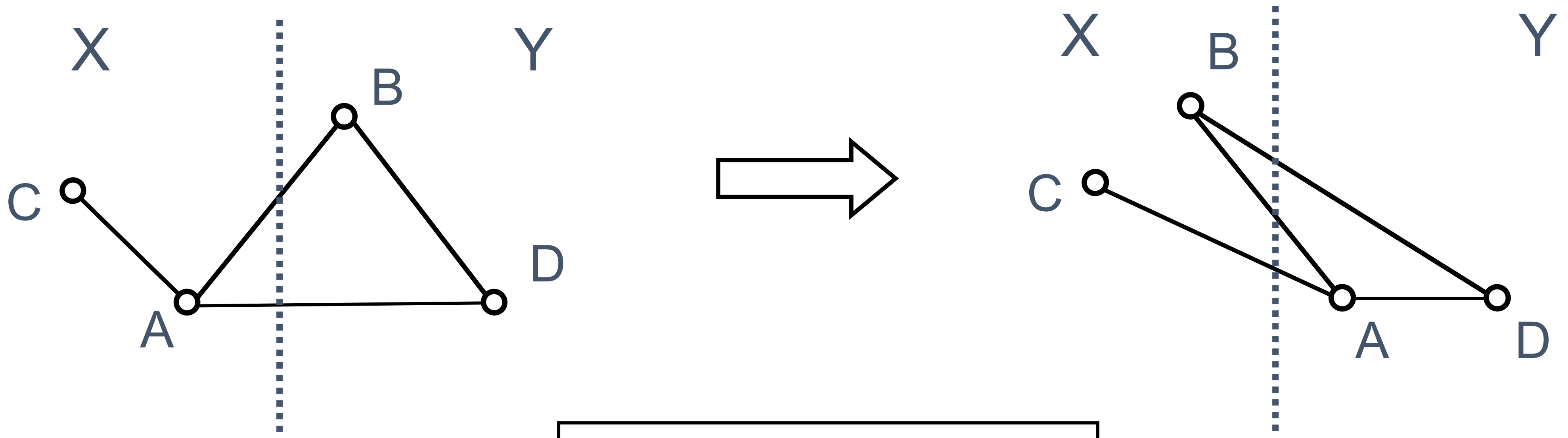
- Note that we want to balance two partitions
- If switch A & B, gain(A,B) = $D_a + D_b$?



gain(A,B) = 1+0 = 1?

Idea of KL Algorithm

- Note that we want to balance two partitions
- If switch A & B, gain(A,B) = $D_A + D_B - 2C_{AB}$
  - c_{AB} : edge cost for AB



gain(A,B) = 1+0-2 = -1

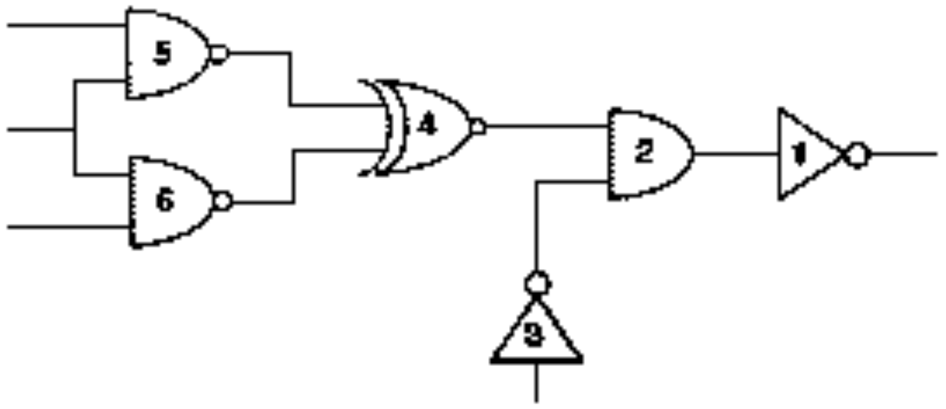
KL Algorithm

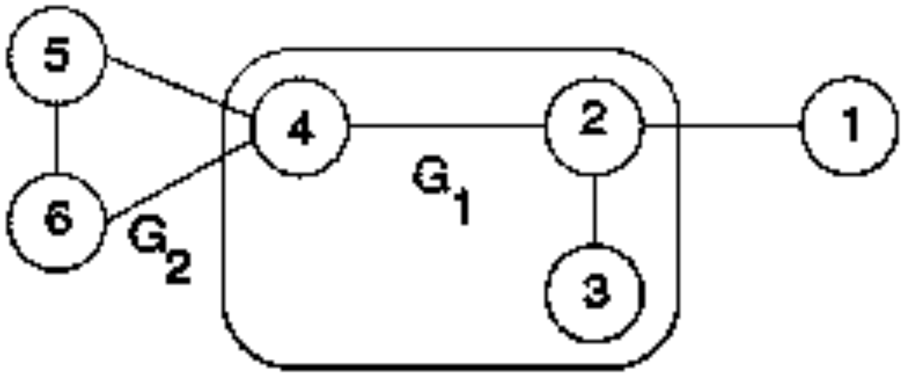
- Start with any initial legal partitions X and Y
- A pass (exchanging each vertex exactly once) is described below:

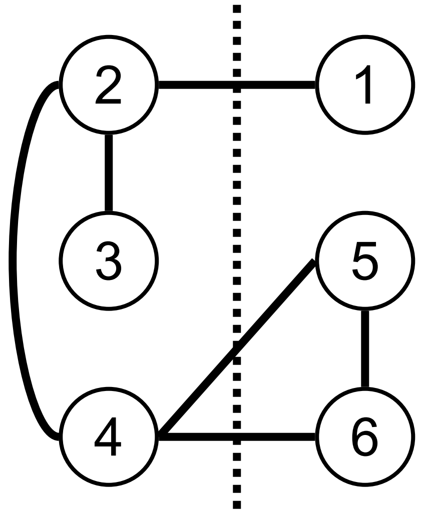
```
1. For i := 1 to n do
       From the unlocked (unexchanged) vertices, choose a pair (A,B)
               s.t. gain(A,B) is largest (greedy)
       Exchange and Lock A, B
       Let $g_i$ = gain(A,B)
2. Find the k s.t. $G=g_1+...+g_k$ is maximized
3. Switch the first k pairs
```

- Repeat the pass until there is no improvement (G=0)

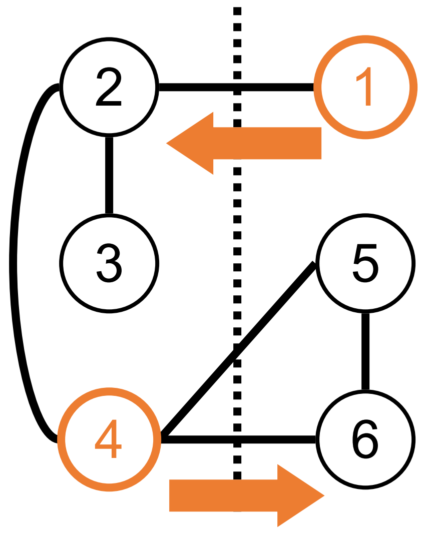
KL Algorithm Walkthrough – 1





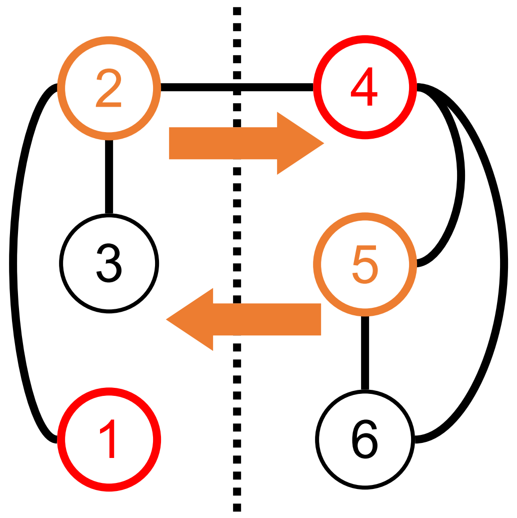


KL Algorithm Walkthrough – 2



- D_1 = 1-0 = 1
- D_2 = 1-2 = -1
- D_3 = 0-1 = -1
- D_4 = 2-1 = 1
- D_5 = 1-1 = 0
- D_6 = 1-1 = 0
- gain(2, 1) = -1 + 1 – 2 = -2
- gain(2, 5) = -1 + 0 – 0 = -1
- gain(2, 6) = -1 + 0 – 0 = -1
- gain(3, 1) = -1 + 1 – 0 = 0
- gain(3, 5) = -1 + 0 – 0 = -1
- gain(3, 6) = -1 + 0 – 0 = -1
- $\boxed{\text{gain(4, 1) = 1 + 1 – 0 = 2}}$ g_1
- gain(4, 5) = 1 + 0 – 2 = -1
- gain(4, 6) = 1 + 0 – 2 = -1

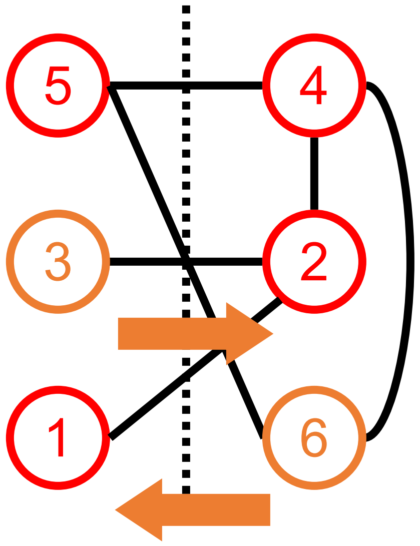
KL Algorithm Walkthrough – 3

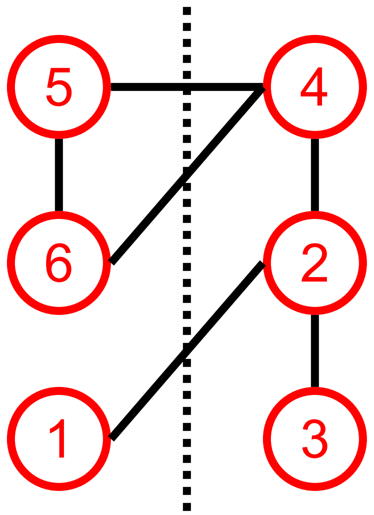


- $D_2 =$ 1 – 2 = -1
- $D_3 =$ 0 – 1 = -1
- $D_5 =$ 0 – 2 = -2
- $D_6 =$ 0 – 2 = -2
- $\boxed{\text{gain(2, 5) = -1 + -2 – 0 = -3}}$ g_2
- gain(2, 6) = -1 + -2 – 0 = -3
- gain(3, 5) = -1 + -2 – 0 = -3
- gain(3, 6) = -1 + -2 – 0 = -3

KL Algorithm Walkthrough – 4

- $D_3 = \quad 1 - 0 = 1$
- $D_6 = \quad 1 - 1 = 0$
- $\boxed{\text{gain}(3, 6) = \quad 1 - 0 - 0 = 1}$ g_3





$g_1 = 2, \quad g_2 = -3, \quad g_3 = 1$

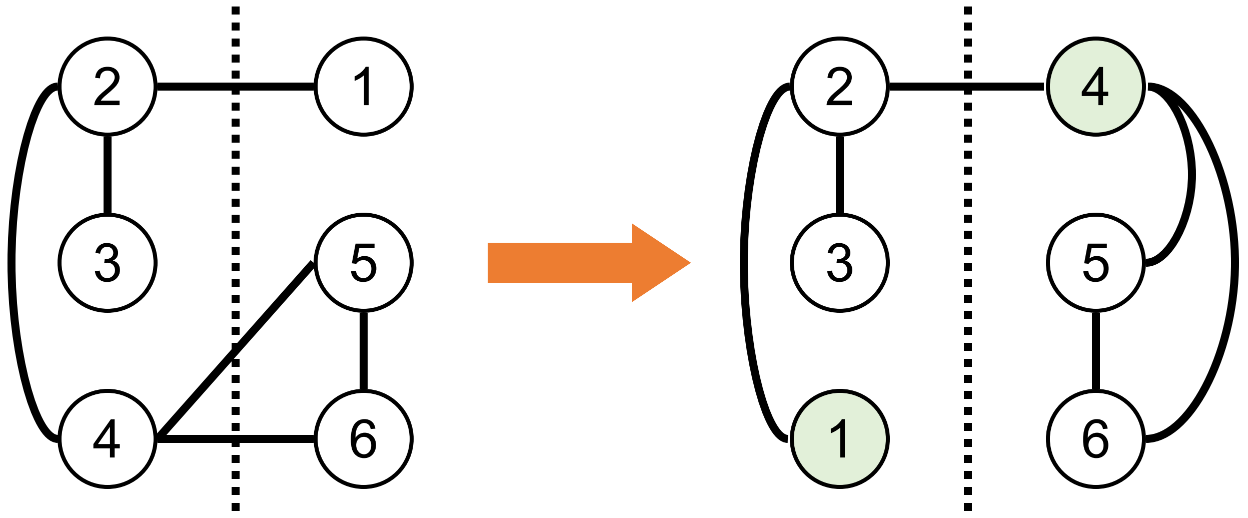
Then ???

We need to find k

- ❑ $\boxed{k = 1: g_1 = \; 2}$
- ❑ $k = 2: g_1 + g_2 = \; -1$
- ❑ $k = 3: g_1 + g_2 + g_3 = 0$

KL Algorithm Walkthrough – 5

- Since k = 1, we only take 1 move → swap (4, 1)



- Finally, we get the result? NO! We only go 1 iteration (i.e., pass)!
- Move on to the next iteration until converged

KL Algorithm Revisited

- Start with any initial legal partitions X and Y
- A pass (exchanging each vertex exactly once) is as follows:

```
1. For i := 1 to n do
       From the unlocked (unexchanged) vertices,
         choose a pair (A,B) s.t. gain(A,B) is largest (greedy)
       Exchange and Lock A, B
       Let $g_i$ = gain(A,B)
2. Find the k s.t. $G=g_1+...+g_k$ is maximized
3. Switch the first k pairs
```

A pass

- Repeat the pass until there is no improvement (G=0)

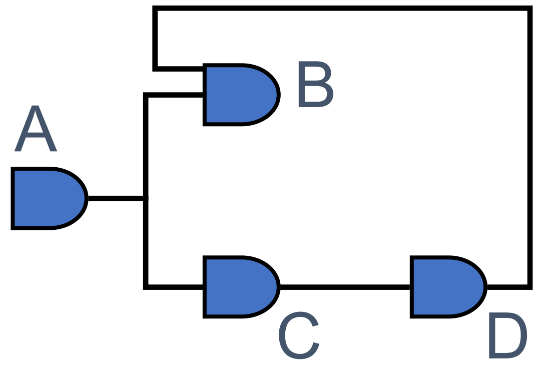
Time Complexity of KL Algorithm

- **For each pass of KL**
  - $O(n^2)$ time to find the best pair to exchange
  - n pairs exchanged
  - Total time is $O(n^3)$ per pass
- **Better implementation can get $O(n^2 \log n)$ time per pass**
  - But requires fairly sophisticated data structure representation
- **Number of passes is usually small**
  - Converged very fast in practice

Drawbacks of KL Algorithm

- **Handle only unit vertex weights**
  - Vertex weights might represent block sizes, different from blocks to blocks in real situation
- **Handle only exact bisection**
  - Need dummy vertices to handle the unbalanced problem
- **Handle only non-hypergraphs**
  - Practical circuits have many terminal nodes for each cell output
  - Need to handle multi-terminal nets directly

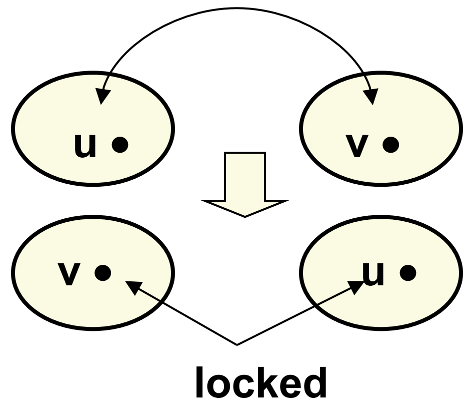
A has two terminals, B and C!



Summary of KL Algorithm

1. Pair-wise exchange of nodes to reduce cut size
2. Allow cut size to increase temporarily within a pass
3. Compute the gain of a swap

```
    Repeat
      Perform a feasible swap of max gain
      Mark swapped nodes “locked”
      Update swap gains
    Until no feasible swap
```



4. Find max prefix partial sum in gain sequence $g_1, g_2, \ldots, g_m$
5. Make corresponding swaps permanent
6. Start another pass if current pass reduces the cut size