Design & Analysis of Physical Design Algorithms

Majid Sarrafzadeh
Elaheh Bozorgzadeh
Ryan Kastner
Ankur Srivatsava

UCLA CS Dept
majid@cs.ucla.edu

ER UCLA

Outline

- Introduction
- Problem Transformation: Upper-Bound
- Practical Implication of Lower-Bound Analysis
- Greedy Algorithms and their proofs
- Greedy vs Global
- Approximation Algorithms
- Probabilistic Algorithms
- Conclusions

- Main message:
  - we need more/better algorithms
  - Better understanding (less hacks)

Introduction

Some points to consider

- Physical Design Problems are getting harder
  - Size, Concurrent optimization, DSM, …
- Novel/effective algorithms are essential in coping with the complexity
  - (mincut vs congestion)
- Analysis of these algorithms are of fundamental importance
  - We can then concentrate on other issues/parameters
- Novel algorithmic paradigms need to be tried

In this talk

- Talk about paradigms that have been (can be) used
  - Upper-bound transformation
- Seemingly unimportant concepts can be very powerful
  - Proof of a greedy algorithm
- There are LOTS of things that we still do not understand (and yet seem important in making progress)
  - congestion
- "We" need to make research progress (abstract/long term in addition to the usual hacks).

Problem Transformation: Upper Bound

Formal Definition

- *A is X(n)-transformable to B:*
  - *The input to A is converted to a suitable input to B*
  - *Problem B is solved*
  - *The output of B is transformed into a correct solution to problem A*
    - *Steps 1 & 3 takes X(n) time*
- *Upper Bound via Transformability:*

If B can be solved in T(n) time and A is X(n)-transformable to B, then A can be solved in T(n) + O(X(n)) time.

- Quality of the solution to A?
  - Bad example: finding min of a list via sorting O(n) transform
  - Good example: element uniqueness via sorting O(n) transform

Multi-way Partitioning Using Bi-partition Heuristics [Wang et al]

Input:

- A target graph to be partitioned.
- k: number of target partitions.

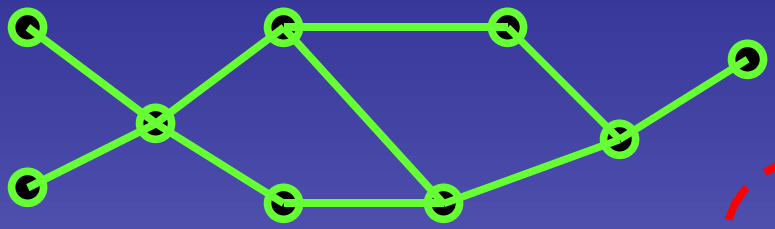
Output:

Each vertex in the target graph gets assigned to one of the target partitions.
Numbers of vertices among target partitions are “the same” (balanced).

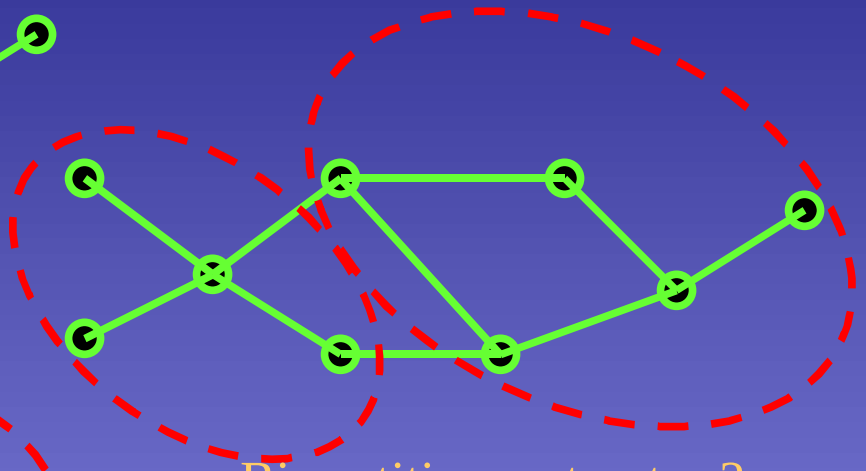
Objective:

The number of edges between target partitions (net-cut) is minimized.

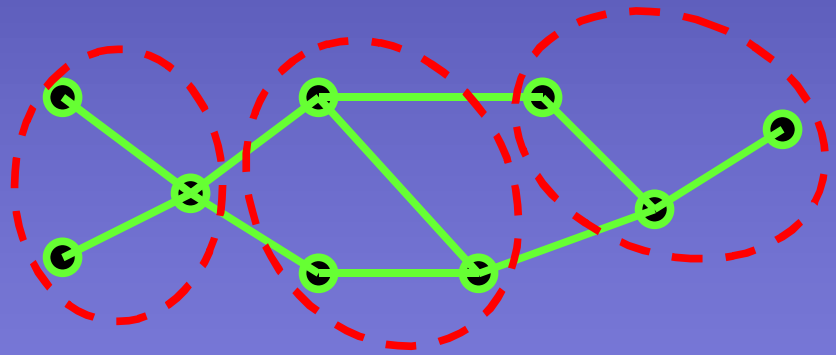
Example



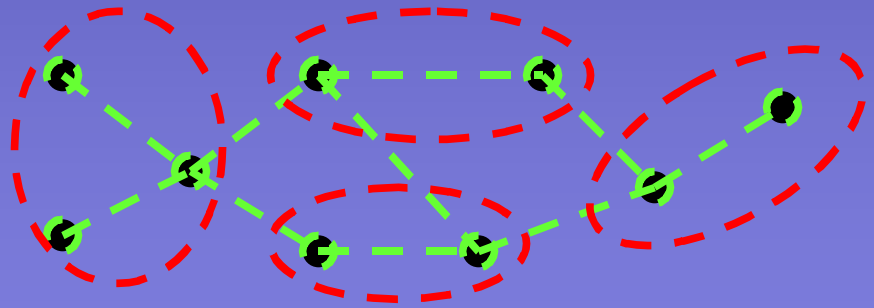
Target Graph



Bi-partition, net-cut = 2

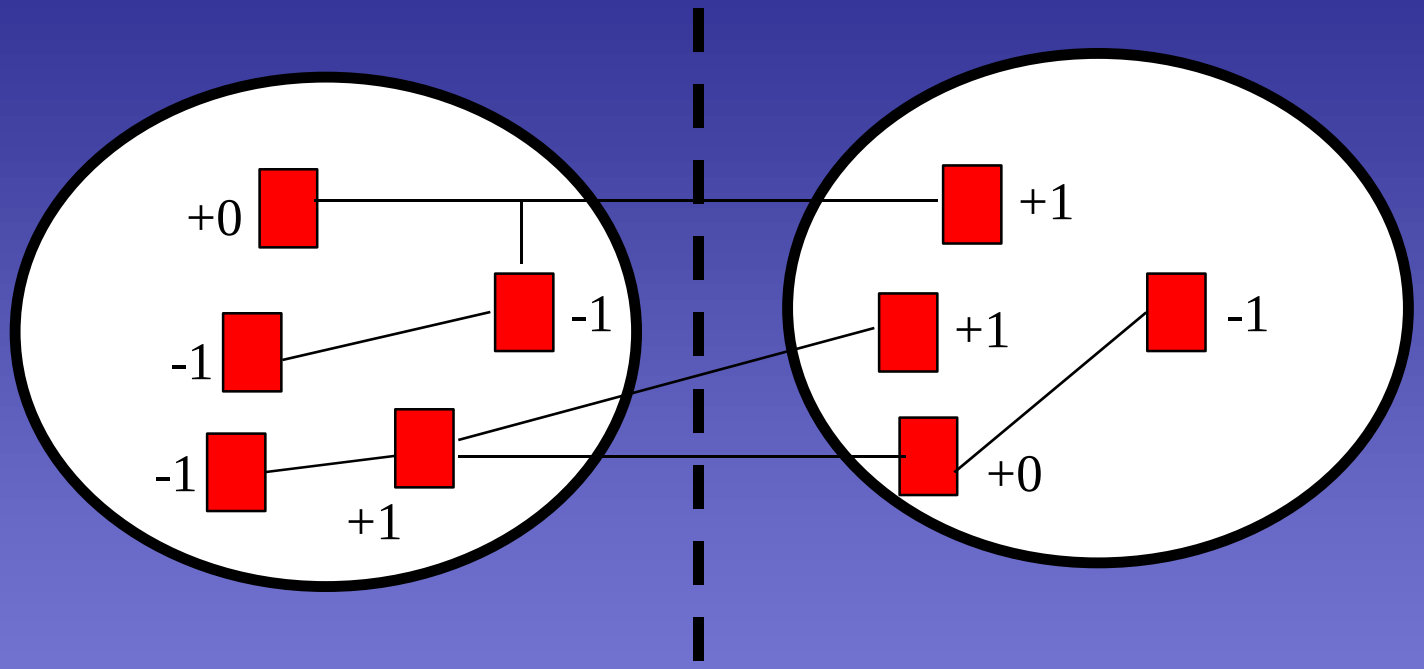


3-way partition, net-cut = 4



4-way partition, net-cut = 5

FM (and FM++): The Industry Standard for Bi-Partitioning



- Each vertex has a “gain” associated with it.
- The vertex with the biggest gain will be moved.

ER UCLA

Possible Approaches For Multi-Way Partitioning

- Direct extension of FM. Each target vertex has k-1 possible moving destinations.
- Using bi-partitioning heuristics (problem transf):
  - Hierarchical approach: Recursively bi-partition the target graph. (1+2+4 bipartition to do 8-way)
  - Flat approach: Iteratively improve the k-way partition result by performing local bi-partitioning (C(8,2) = 28 or more bipartition to do 8-way)
  - Question: which one is more powerful?

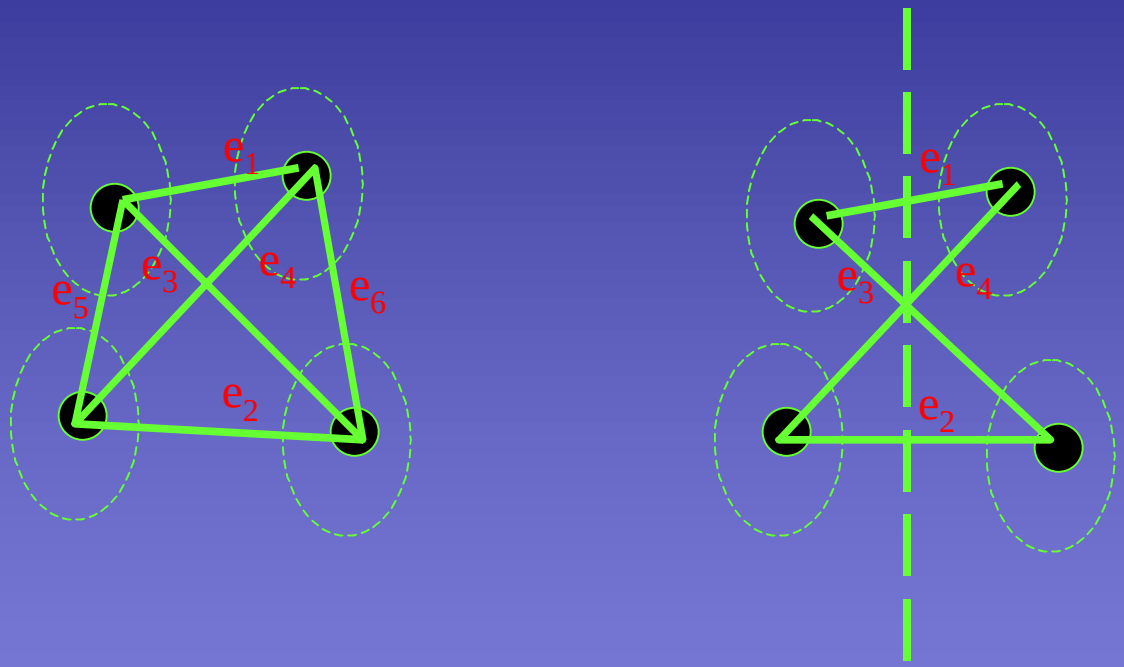
Why problem transformation?

- The direct extension of FM approach does not yield good partitioning results in general.
- The state-of-the-art bi-partitioning tools are very effective.
- It is straightforward (little R&D) to solve multi-way partitioning problem using existing bi-partitioning tools via hierarchical or flat approaches.

Bi-partition Algorithms

- A δ-approximation algorithm: the bi-partition result is less than or equal to δC_{opt}.
- An α-balanced bi-section problem: the number of nodes in each partition is between αn and $(1-\alpha)n$. (A perfectly balanced bi-section problem is a 0.5-balanced problem.)

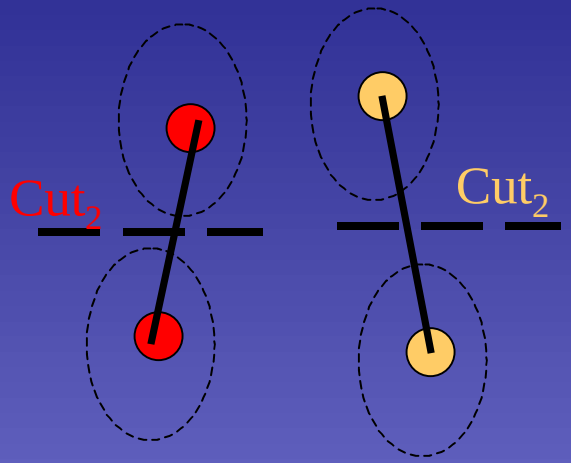
The First Cut

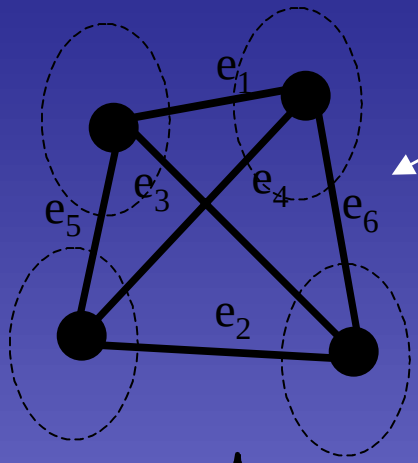


$$Cut_1 = e_1 + e_2 + e_3 + e_4 <= e_1 + e_2 + e_3 + e_4 + e_5 + e_6 <= \delta\ OPT$$

ER UCLA

The Second Cut



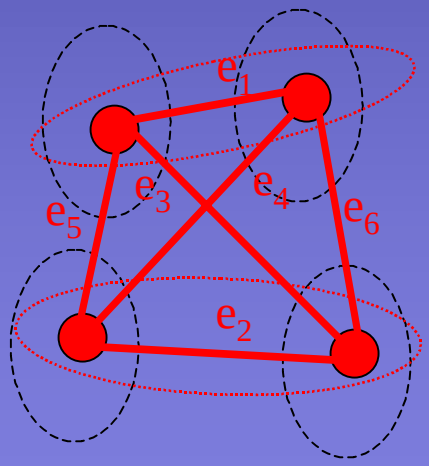


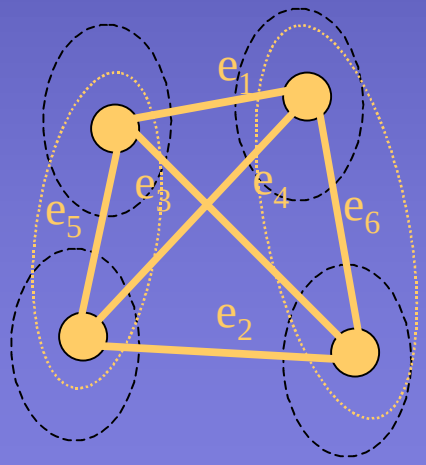
Optimal solution

$Cut_2 <= \delta (e_1 + e_2 + e_3 + e_4 + e_5 + e_6)$

$Cut_2 <= \delta (e_1 + e_2 + e_3 + e_4 + e_5 + e_6)$

$Cut_2 <= \delta OPT$





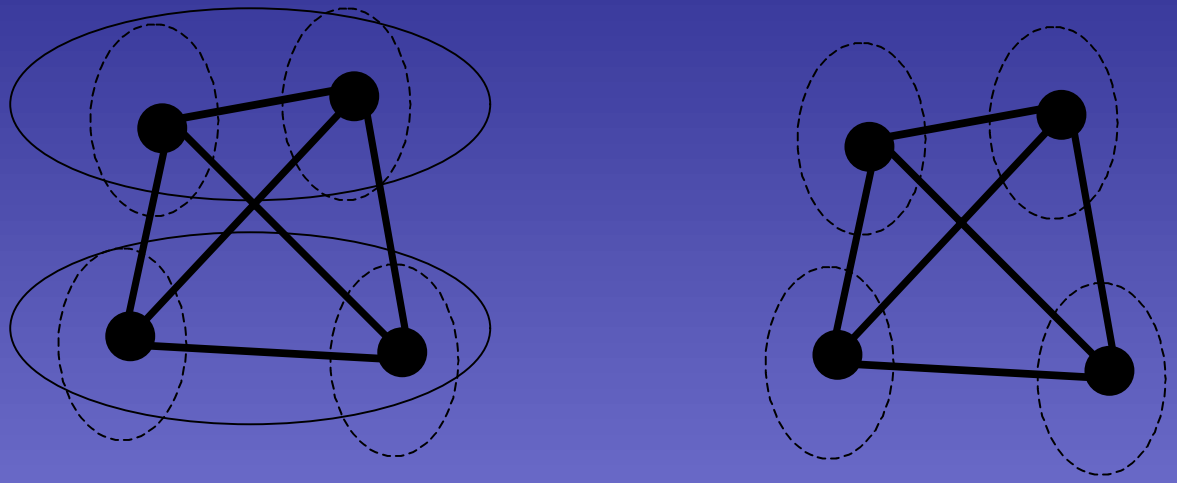
ER UCLA

Conclusion

- For hierarchical approach:

$$C_{hie} = O(\delta \log k) C_{opt}$$

The Flat Approach



$$C_{hie} = O(\delta k) C_{opt}$$

Variations of Flat Approaches

- Random: randomly pick two partitions at a time.
- Exhaustive: use a specific sequence to exhaustively pick all possible paris of partitions.
- Cut-based: pick a pair or two most tightly or loosely connected partitions.
- Gain-based: pick a pair of two partitions between which the cutsize reduction is maximum or minimum during last pass.

Multi-pass Flat vs. Hierarchical

circuit	best of allway		hierarchical		hie. vs. all-way	
	net-cut	runtime(s)	net-cut	runtime(s)	%improv.	speedup
fract	64	23.5	56	1.21	12.5%	19.4
struct	133	627.6	128	5.74	3.8%	109
p1	145	411.7	150	5.17	-3.4%	79.6
p2	455	1979	450	12.5	1.1%	158
biomed	305	1995	213	13.18	30.2%	151
in1	203	1249	175	9.68	13.8%	129
in2	979	5006	898	44.9	8.3%	111
in3	2210	5512	2070	80.1	6.3%	68.8
avqs	532	4485	532	42.6	0%	105
avql	532	24315	543	47.8	-2.1%	509
avg.					7.1%	144

The hierarchical approach is more effective than the all-way approach.

Conclusion

- The hierarchical approach is the correct way to solve the multi-way partition problem using a state-of-the-art bi-partitioner.
- Problem transformation can be very powerful (or very weak)
- A little analysis goes a long way (and results can be counter intuitive)

Practical Implications of Lower Bound Analysis

Formal Definition

- *A is X(n)-transformable to B:*
  - *The input to A is converted to a suitable input to B*
  - *Problem B is solved*
  - *The output of B is transformed into a correct solution to problem a*

- *LOWER Bound via Transformability:*

If A is known to require T(n) time and A is X(n)-transformable to B, then B requires at least T(n) - O(X(n)) time.

Proofs of NP-Completeness are important because

- Knowledge of NP-Completeness points out that research must be directed towards finding efficient heuristics and not optimal algorithms

- **The proof itself gives important information about the problem. It can give useful directions to algorithm designers.**

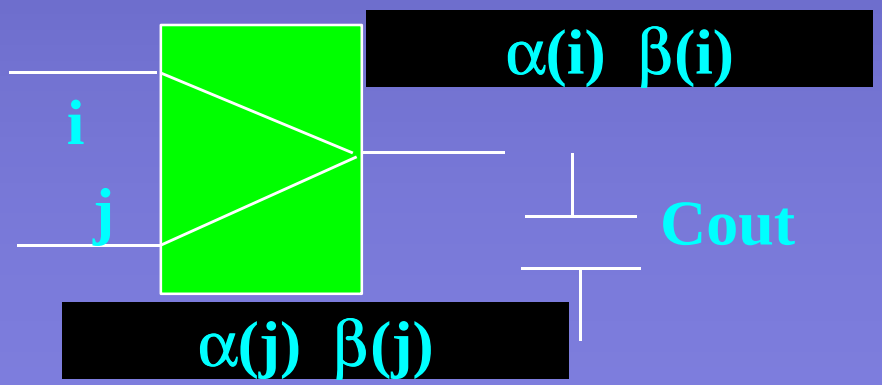
Case Study: Global Buffer Insertion

- Problem NP-Complete in the Load Dependent Delay Model

$$\delta(i) = \alpha(i) + \beta(i)^* \text{ Cout}$$

$\alpha(i)$: Internal Delay from pin i to output

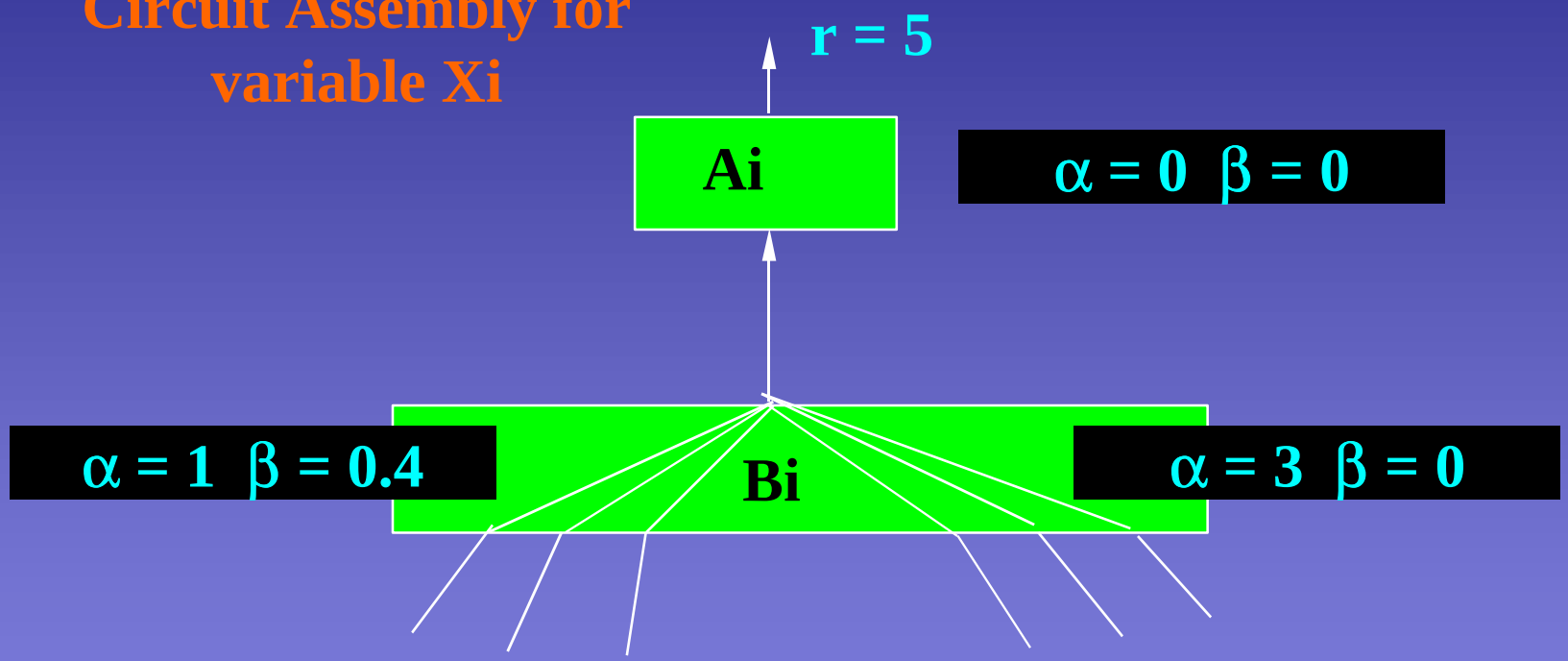
$\beta(i)$: Load Sensitivity



Transforming 3SAT to Buffer Insertion

3SAT Problem : C = (Xi + Xj + Xk). (Xi' + Xl + Xm')……..

Circuit Assembly for variable Xi



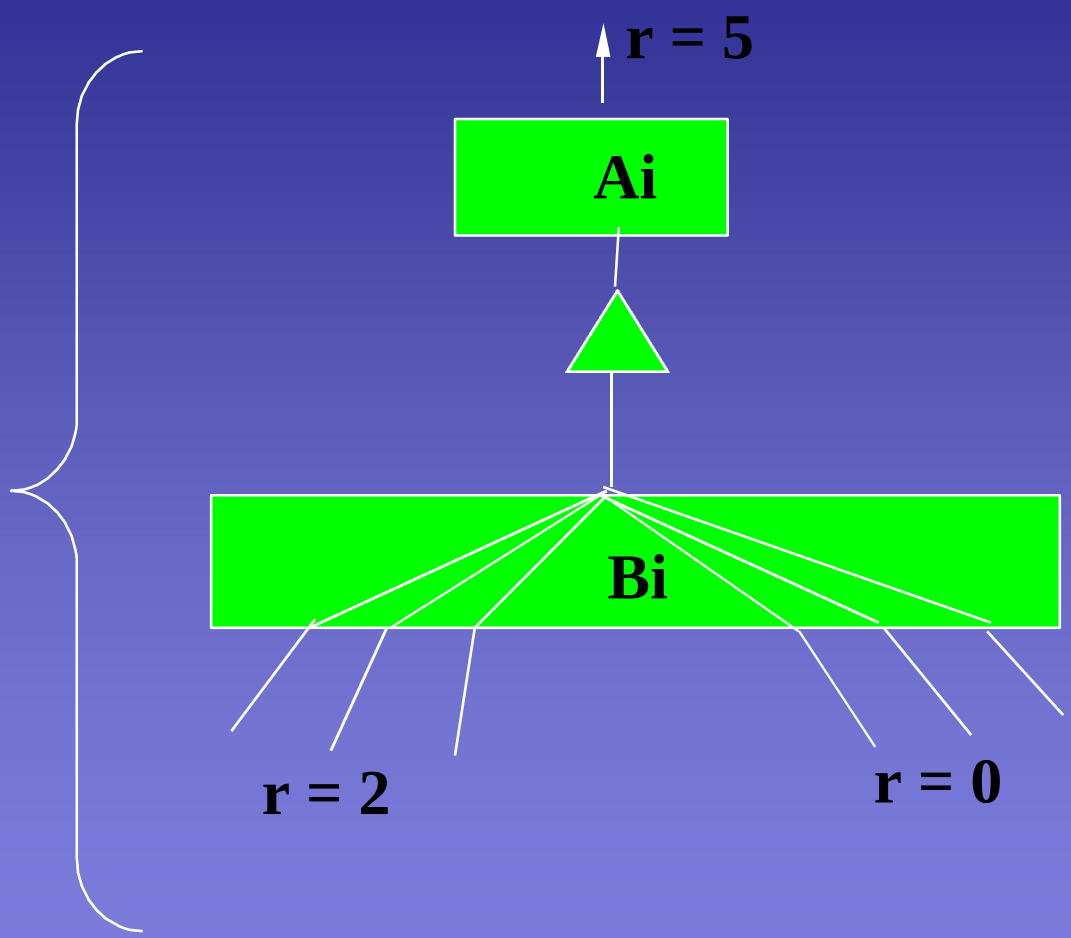
Connected to Clauses Where Xi appears as positive phase

Connected to Clauses Where Xi appears as negative phase

ER UCLA

Transforming 3SAT.....

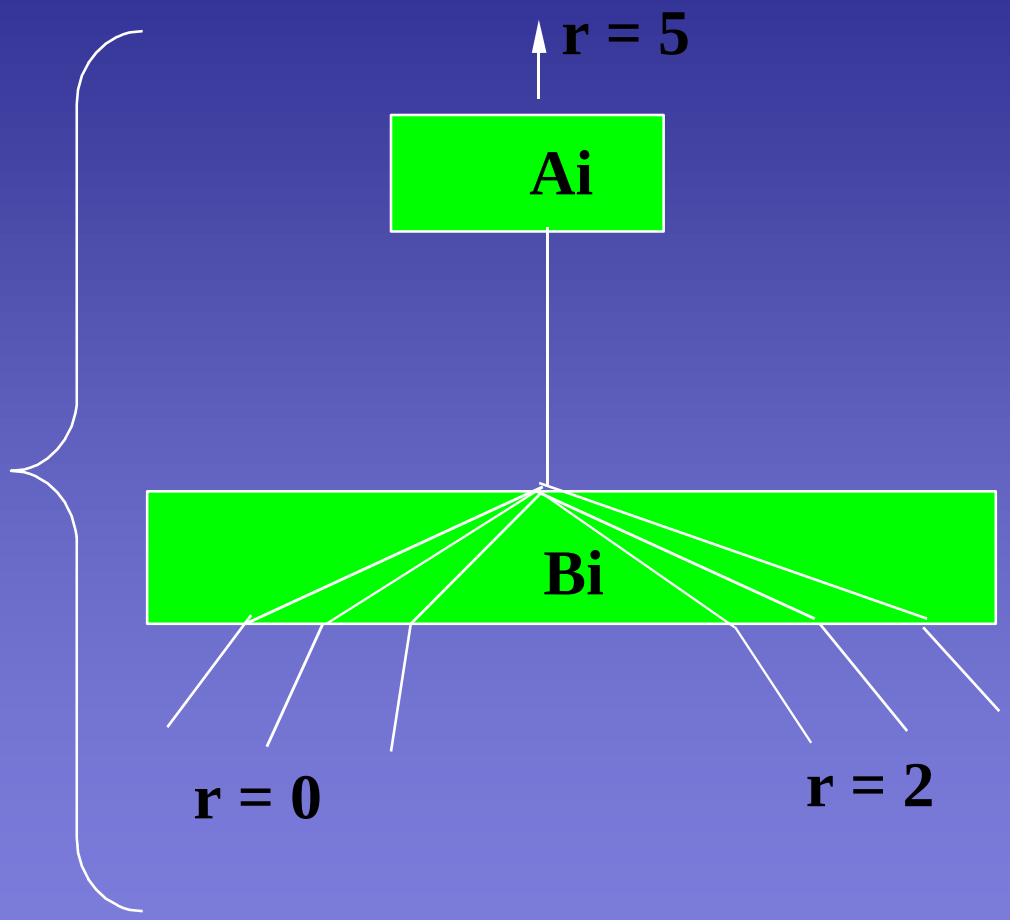
If the net BiAi is buffered then the required time at one set of fanins is 2 and at other set of fanins is 0



ER UCLA

Transforming 3SAT.....

If the net BiAi is not buffered then the required time behavior becomes opposite



ER UCLA

Observations from Transformation

- Buffer Insertion on net BiAi helped one set of fanins and was not beneficial for the other. No buffer insertion had the opposite effect.
- This happens primarily because the load sensitivity (β) of gate Bi is different for different sets of input pins
- The problem became NP-Complete primarily because the load sensitivity (β) for different input pins of a gate is different

Observation from Transformation

- Since each input pin has different load sensitivity, an optimal buffering solution w.r.t. a particular input pin of the net connected at the output of the gate, may not be optimal w.r.t. other pins. Hence the problem is NP-Complete

All these observations were made directly from the transformation and the proof of NP-Completeness

Practical Implications

- If all gates of a circuit have the same load sensitivity then the problem is optimally solvable (if the local problem is optimally solvable).
- Fanout Optimization on substructures which have similar value of β will be more effective
- This also suggests methodologies for partitioning the circuit

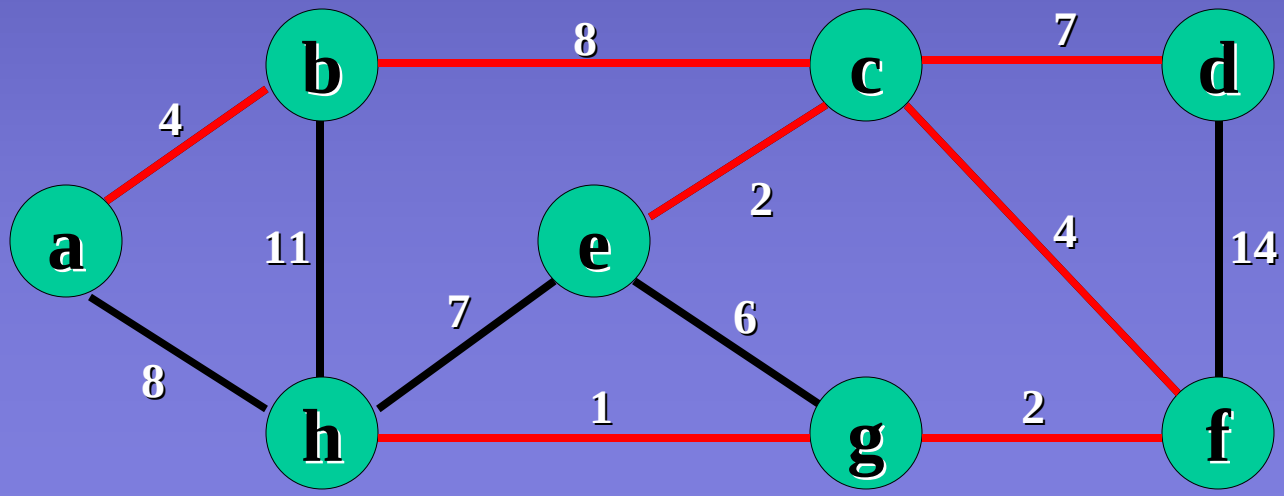
Conclusion

- Proofs of NP-Completeness can give a deep insight into the optimization problem. They also assist algorithm designers in finding useful ways of solving the problem.
- Who will prove these things?

Proof of Greedy Algorithms

Greedy Algorithms

- Make a locally optimal choice to achieve a globally optimal solution
- Correct greedy algorithm often the simplest
- Example: Prim's Minimum Spanning Tree algorithm
  - Iteratively grow the spanning tree
  - Select edge which adds minimum value at each step

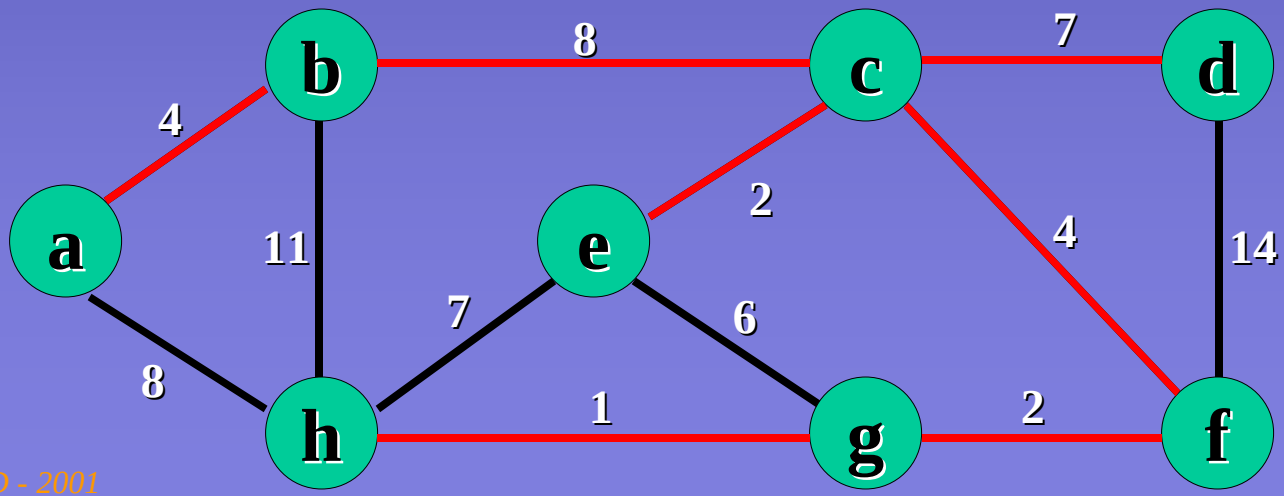


ER UCLA

Proof of Greedy Algorithm Correctness

Correct greedy algorithms exhibit two properties:

- Optimal substructure
  - Optimal solution to the problem contains optimal solutions to sub-problems
  - Stopping Prim's MST algorithm at any instant gives optimal MST for the currently "spanned" nodes
- Greedy choice property
  - Globally optimal solution can be built using a series of locally optimal choices



Optimal Substructure

MST for {a,b,c}

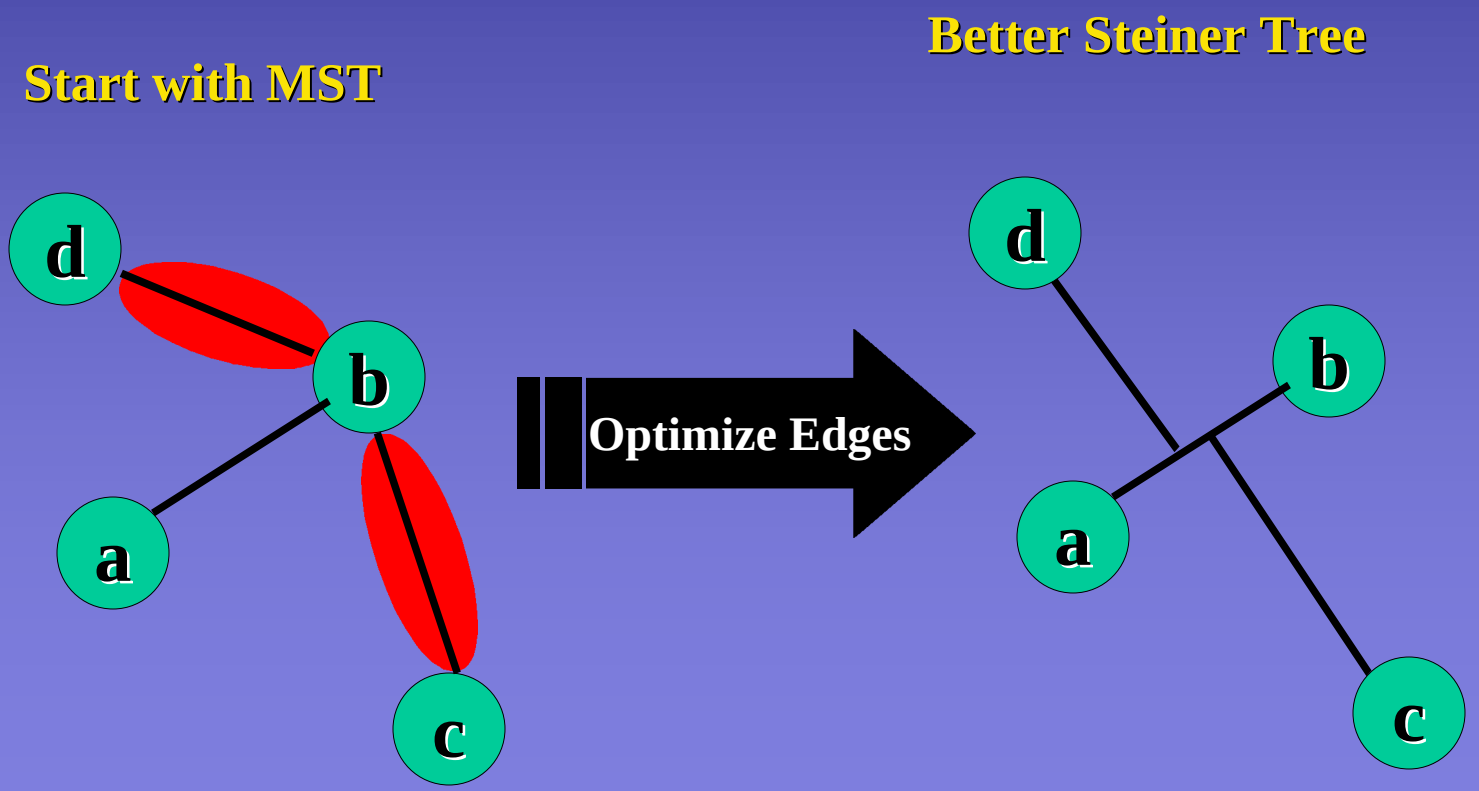
Greedy Choice

Next edge is minimum wieghted connected edge (c,e)

ER UCLA

Greedily Solving NP-Complete Problems

- Use greedy solution as an optimization starting point
  Add more complex optimizations to greedy algorithm



ER
UCLA

Use of Greedy Algorithms

- Simple
- Analysis may reveal interesting properties
- Can be used as heuristics in solving related problems

Greedy vs Global

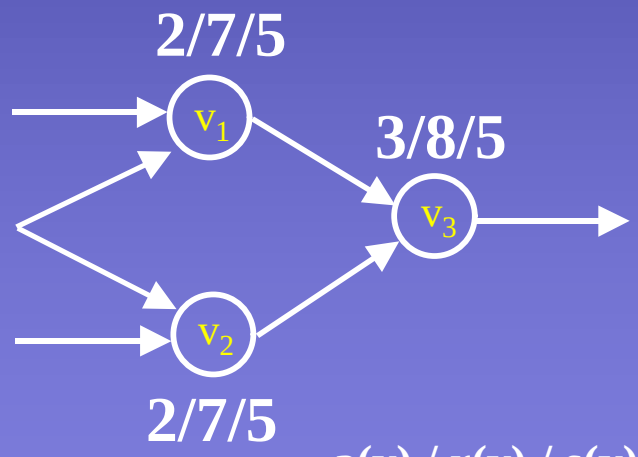
Slack Assignment Problem

- Combinational network ⟶ Directed-acyclic graph
- Given a delay model and timing requirement, calculate node slack: $s(v) = r(v) - a(v), \quad v \in \mathbf{V}$
- Slack distribution: $\mathbf{S(V)} = [s(v_1), s(v_2), \ldots, s(v_n)]$
  total slack: $|\mathbf{S(V)}| = \Sigma\, s(v_i)$
- Slack assignment: assign incremental delays
  $\Delta\mathbf{D(V)} = [\Delta d(v_1), \Delta d(v_2), \ldots, \Delta d(v_n)]$ to $\mathbf{V}$

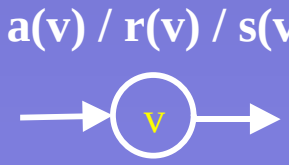
  If $\mathbf{S}_{\Delta}(\mathbf{V}) \geq 0$, the slack assignment is effective and $|\Delta\mathbf{D(V)}|$ is called effective slack.

Slack Assignment Problem

- Potential slack: maximum effective slack $|\Delta \mathbf{D}_m(\mathbf{V})|$
- A slack assignment is (global/) optimal if it leads to potential slack.



Unit delay model

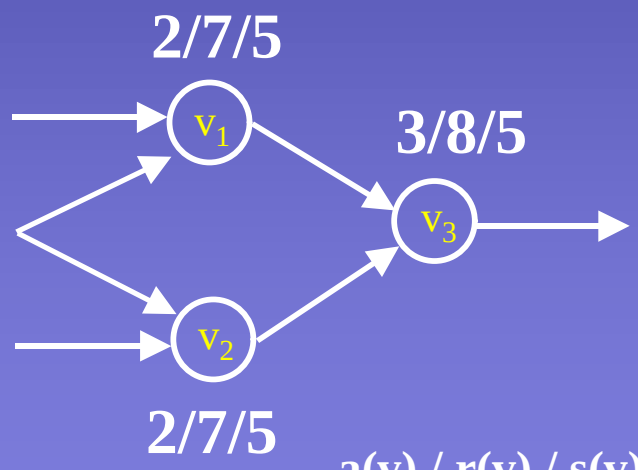


- Slack distribution $\mathbf{S}(\mathbf{V})$ = [5 5 5]
- $\Delta \mathbf{D}(\mathbf{V})$ = [2 1 1] is a slack assignment
- $\Delta \mathbf{D}_m(\mathbf{V})$ = [5 5 0] is a better slack assignment
- Potential slack = $|\Delta \mathbf{D}_m(\mathbf{V})|$ = 10

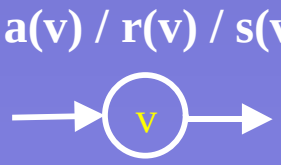
ER UCLA

ZSA: Greedy

- Slack represents an upper bound of delay increase while keeping timing performance
- Zero-Slack Algorithm (ZSA) [Nair et al, 1989]:



Unit delay model

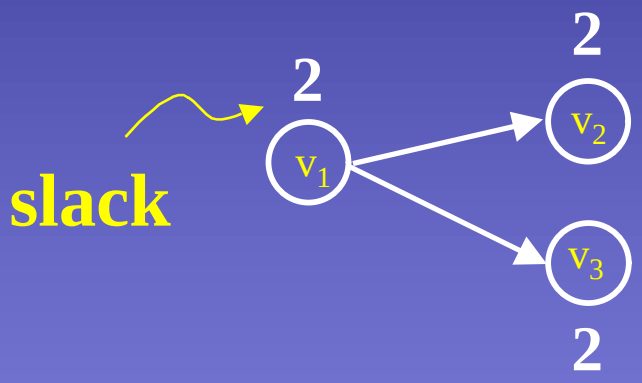


- Assign $\Delta d = 2.5$ to v_1 and v_3 each
- Slack distribution becomes [0 2.5 0]
- Assign $\Delta d = 2.5$ to v_2
- We have slack assignment [2.5 2.5 2.5] with effective slack of 7.5

ER UCLA

Potential Slack (PS): Basic Idea of Finding PS

- Try to identify a maximal-independent set to increase slack budget (effective slack):



--- Select v_2 and v_3 for slack assignment since v_2 and v_3 are independent nodes

--- Potential slack = 4

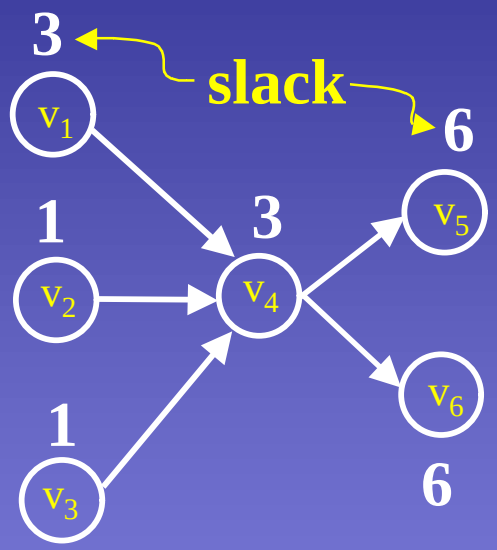
- Study slack sensitivity to reduce slack penalty:

Penalty --- When Δd is assigned to node v, the slacks of v's neighbors may be reduced.

ER UCLA

Potential Slack (PS): Basic Idea of Finding PS

An example of slack sensitivity



--- Slacks of nodes v_5 and v_6 are larger than those of v_1 ~ v_4 , the former being sensitive to the latter

--- Order of nodes for slack assignment is important

--- First selecting v_5 and v_6 gives better result

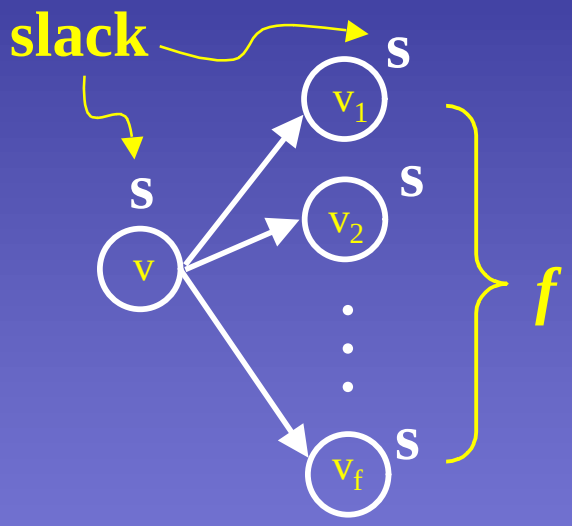
Two different slack assignments:

$\Delta_1\mathbf{D}$ = [3 1 1 0 3 3] , $|\Delta_1\mathbf{D}|$ = 11

$\Delta_2\mathbf{D}$ = [1 1 1 0 5 5] , $|\Delta_2\mathbf{D}|$ = 13

ER UCLA

Potential Slack (PS): A typical example



- ZSA: $|\Delta_1 \mathbf{D(V)}| = (f + 1)\, s / 2$
- Optimal: $|\Delta_m \mathbf{D(V)}| = f\, s$
- $|\Delta_m \mathbf{D(V)}| \approx 2\, |\Delta_1 \mathbf{D(V)}|$ for $f >> 1$
- Total slack = $(f + 1)\, s$

ER UCLA

Algorithm: theoretical results

Theorem 1 [Chen-Sarrafzadeh, ICCAD 2000):

Maximal-Independent-Set based Algorithm (MISA) generates an optimal slack assignment.

Theorem 2:

The time complexity of MISA is $O(n^3)$, where n is the number of nodes in a given graph.

(if too complex, build heuristics around it)

Application: gate sizing problem

- Use slack assignment result to select gates to be down-sized.
- High potential slack promises significant area/power reduction.

Application: placement problem

- Potential slack (PS) can be translated into the freedom/flexibility of all signal nets during physical design phase. The more potential slack, the easier to route signal nets without violating the timing constraints.
-
- Potential slack based results are very effective

Conclusions

- Global algorithms are harder to design and analyze, have higher time complexity, however can produce good (optimal?) results.

- Is it worth it? Deep understanding on the structures of the solution
  - What if too complex to implement?
  - What if takes too much time to run?

Approximation Algorithms

Approximation Algorithm

- Problem : *P*
- Objective : minimize *f* (*f* : the optimal value*)
- Instance *I* ¸ *P*
- α-Approximation Algorithm *A* (Heuristic) for *P*

$$f(I,A) \leq \alpha . f^{*}(I)$$

Approximation Algorithm

- How to Evaluate Quality of Heuristics?
  1. Analyzing the data obtained by running on set of benchmark
     → Only comparison among set of samples!
  2. Deriving Lower Bounds or limits of approximability using approximation analysis methods
     → Not Easy to Analyze Lower Bounds!

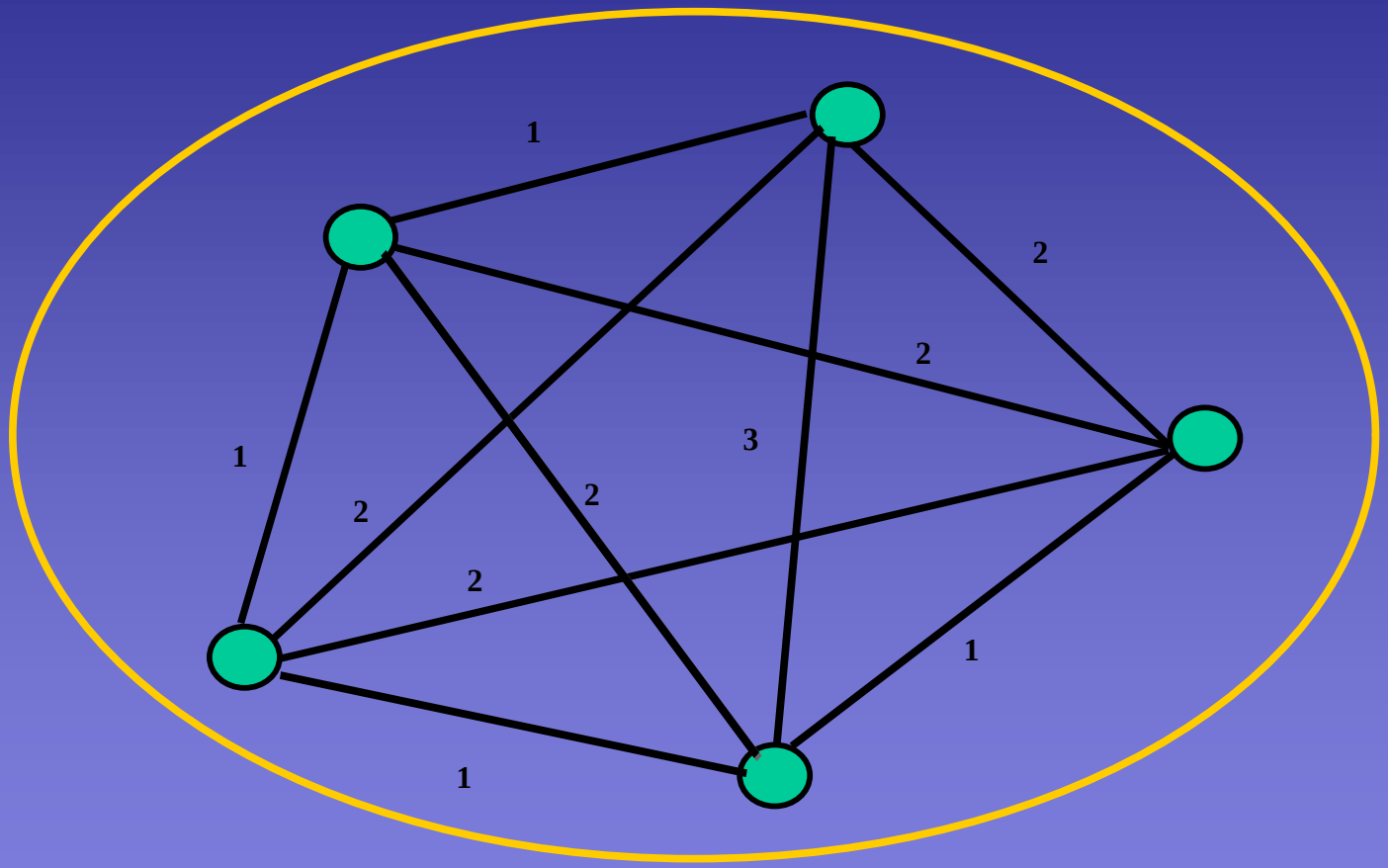
- What can analyzing performance of an algorithm tell us?
  - If the approximation is good enough
  - Approximation algorithms bound the search space

Example: Clustering Algorithm

- Problem:
  - Given an undirected weighted graph ***G***, objective function ***f***, and integer ***k***,
  - Partition the graph into ***k*** clusters such that value of ***f*** is minimized.
- α-Approximation Algorithm:
  - Proposed by Gonzalez (1985)
  - Assumptions:
    - ***G*** is Complete Graph
    - ***f*** is to minimize the maximum weight of an edge inside a cluster
    - Weights of edges satisfy triangle inequality
  - Complexity: $O(k \cdot n)$
    - n is number of nodes in graph G

Example: Clustering Algorithm

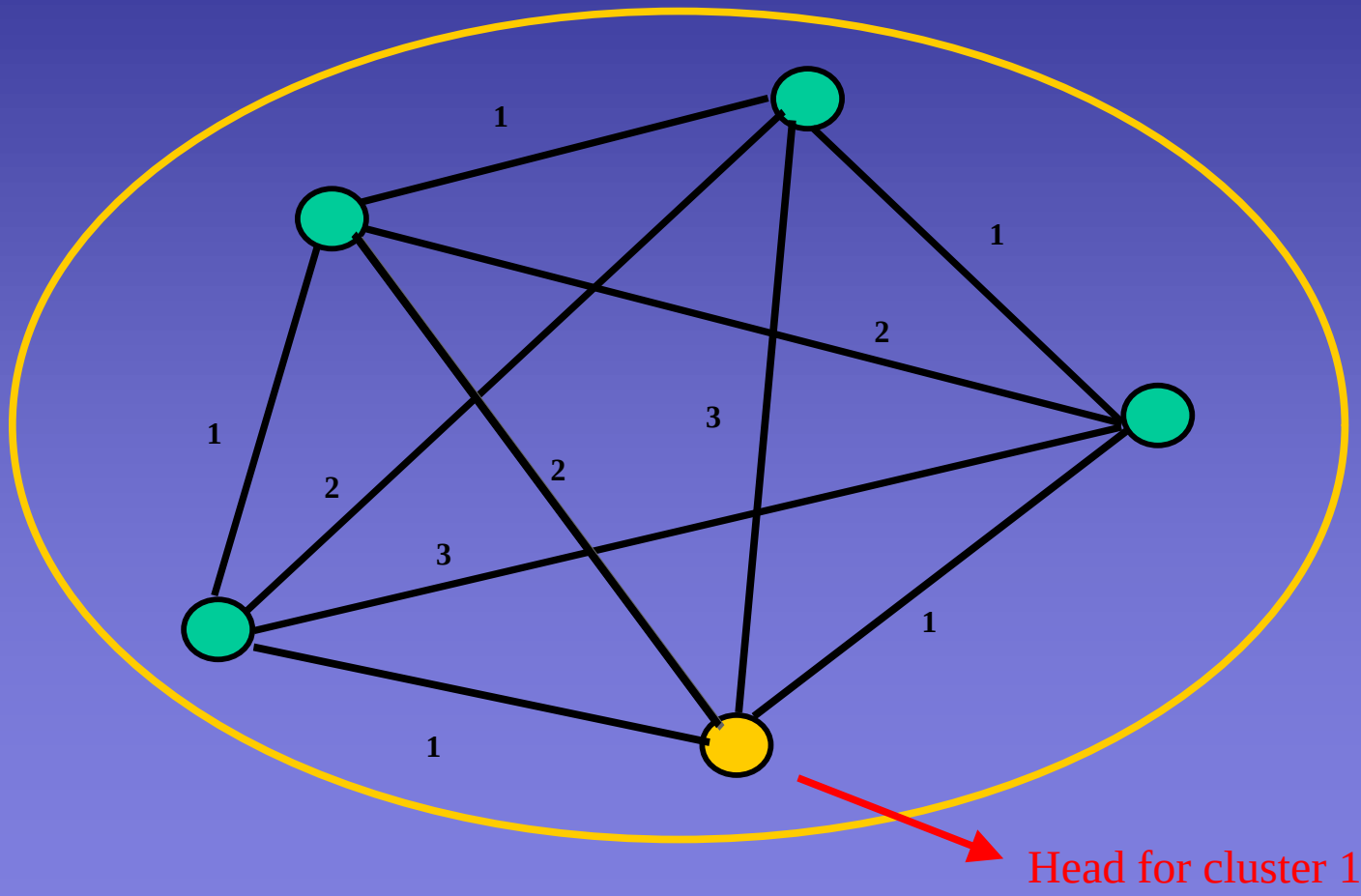
- Generate two clusters on complete graph G (k=2).



ER UCLA

Example: Clustering Algorithm

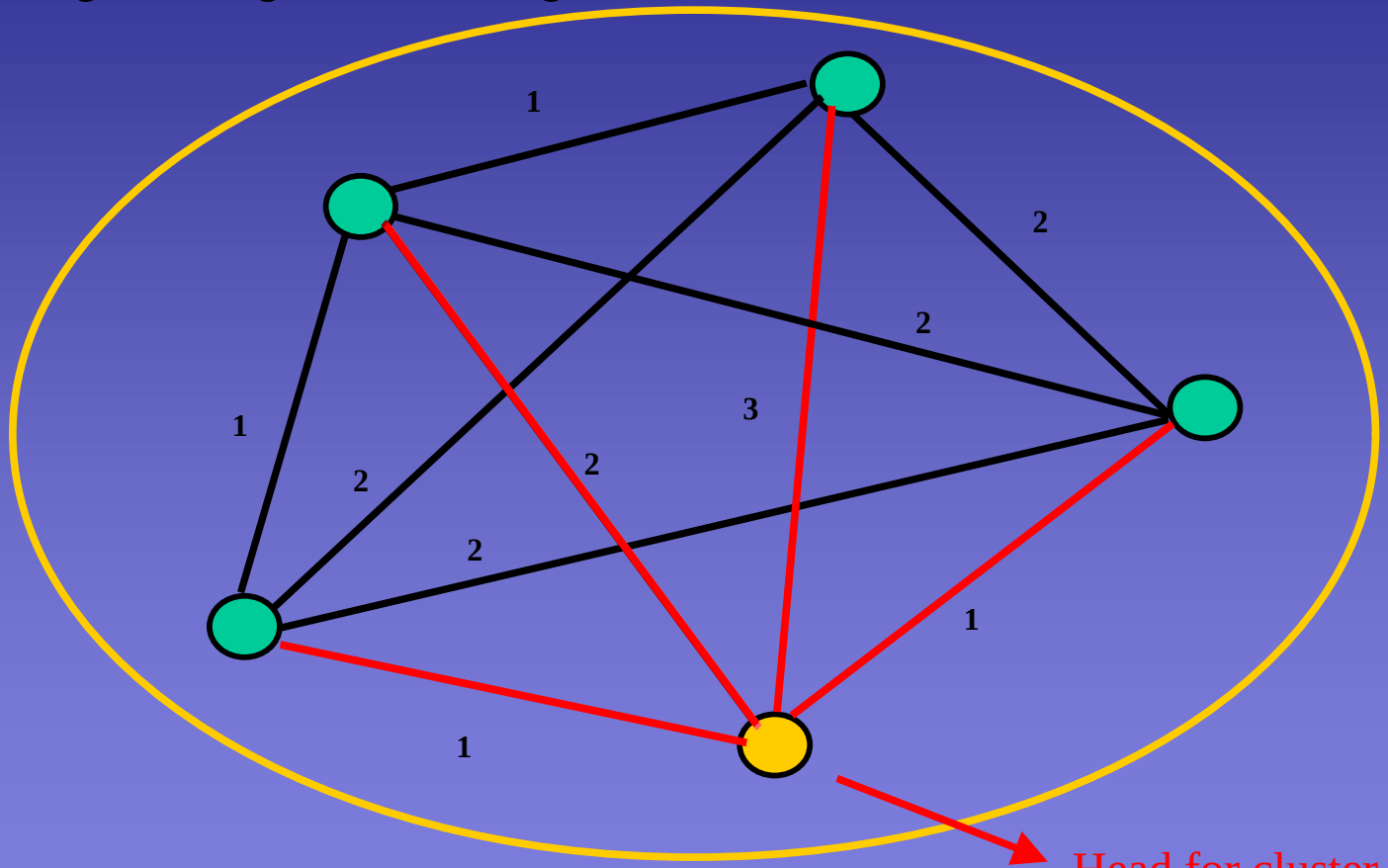
- Generate a cluster including all the nodes.
- Pick an arbitrary node as head of current cluster.



ER UCLA

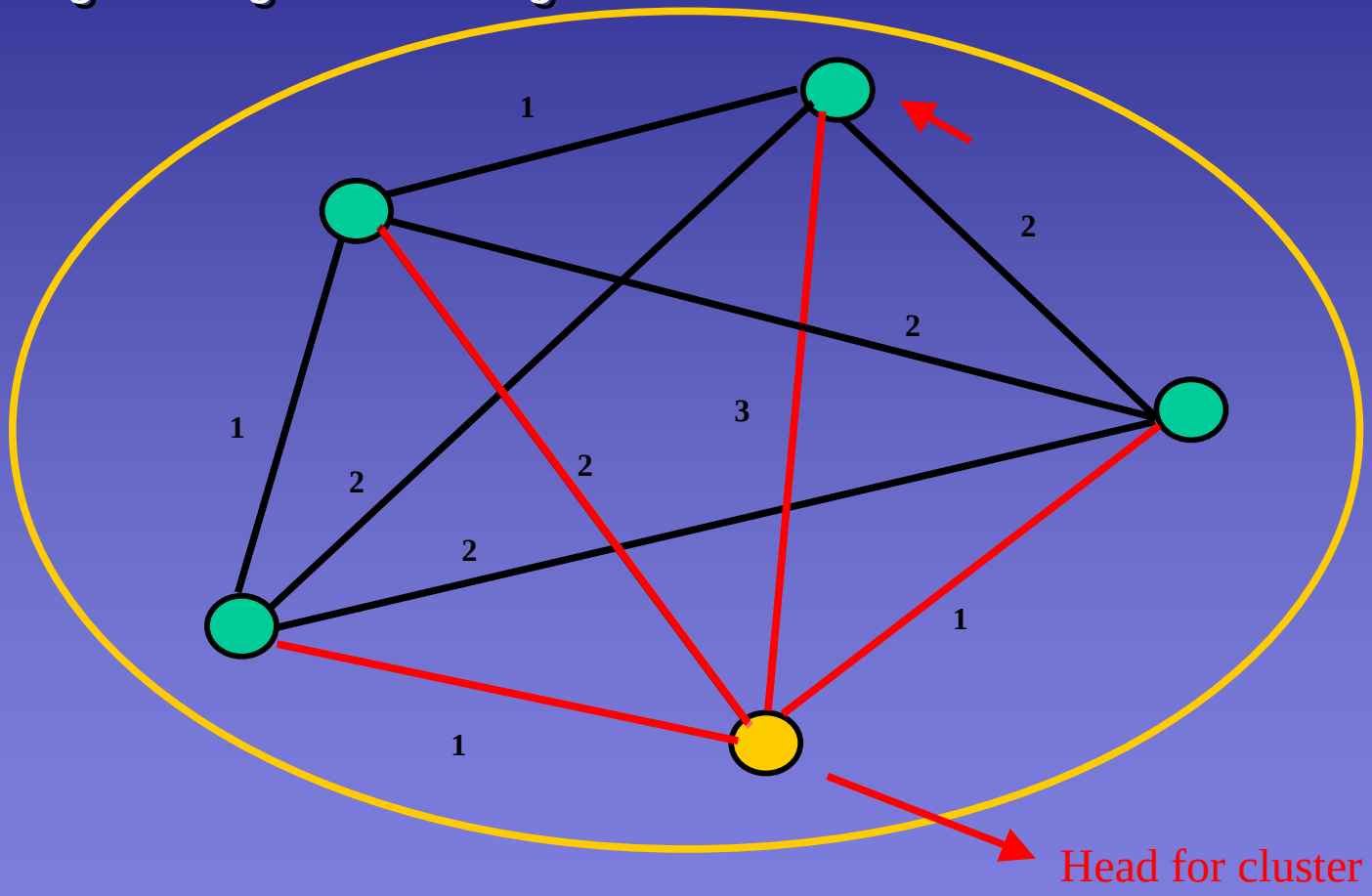
Example: Clustering Algorithm

- Select the node that is connected to the head of cluster through the edge with greatest weight



Example: Clustering Algorithm

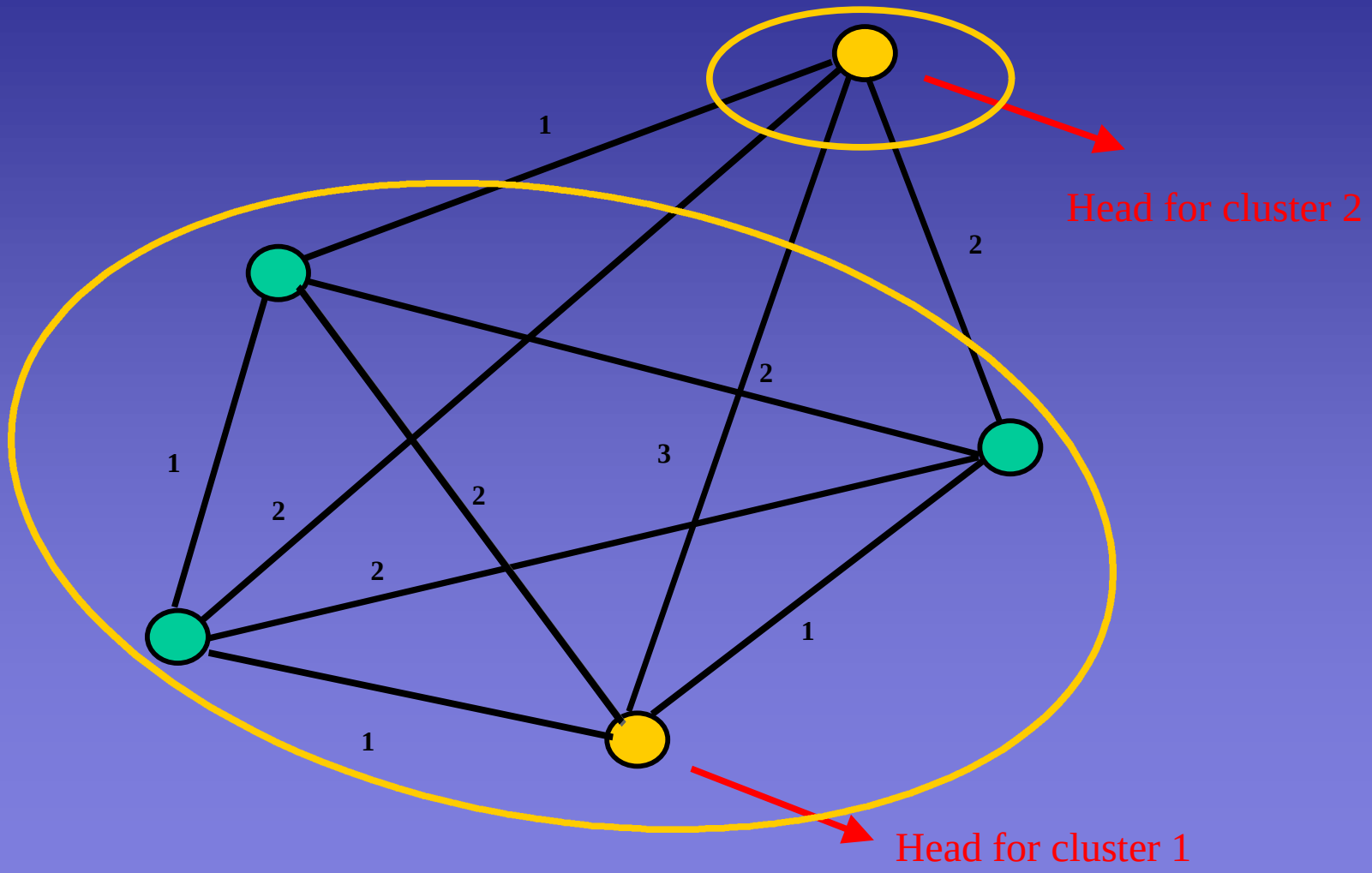
- Select the node that is connected to head of its cluster through the edge with greatest weight.



ER UCLA

Example: Clustering Algorithm

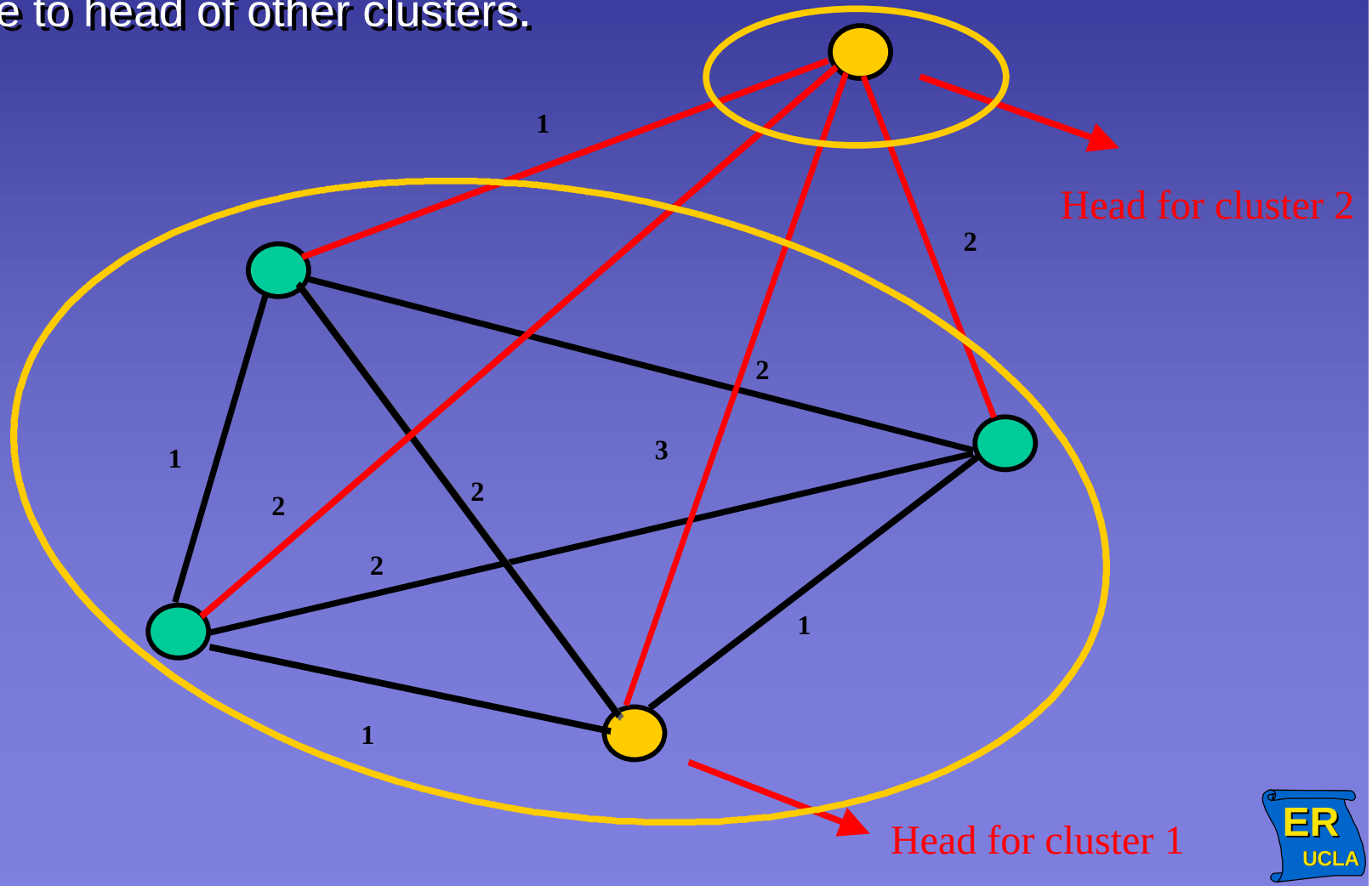
- Separate the node, being the head of new cluster.



ER UCLA

Example: Clustering Algorithm

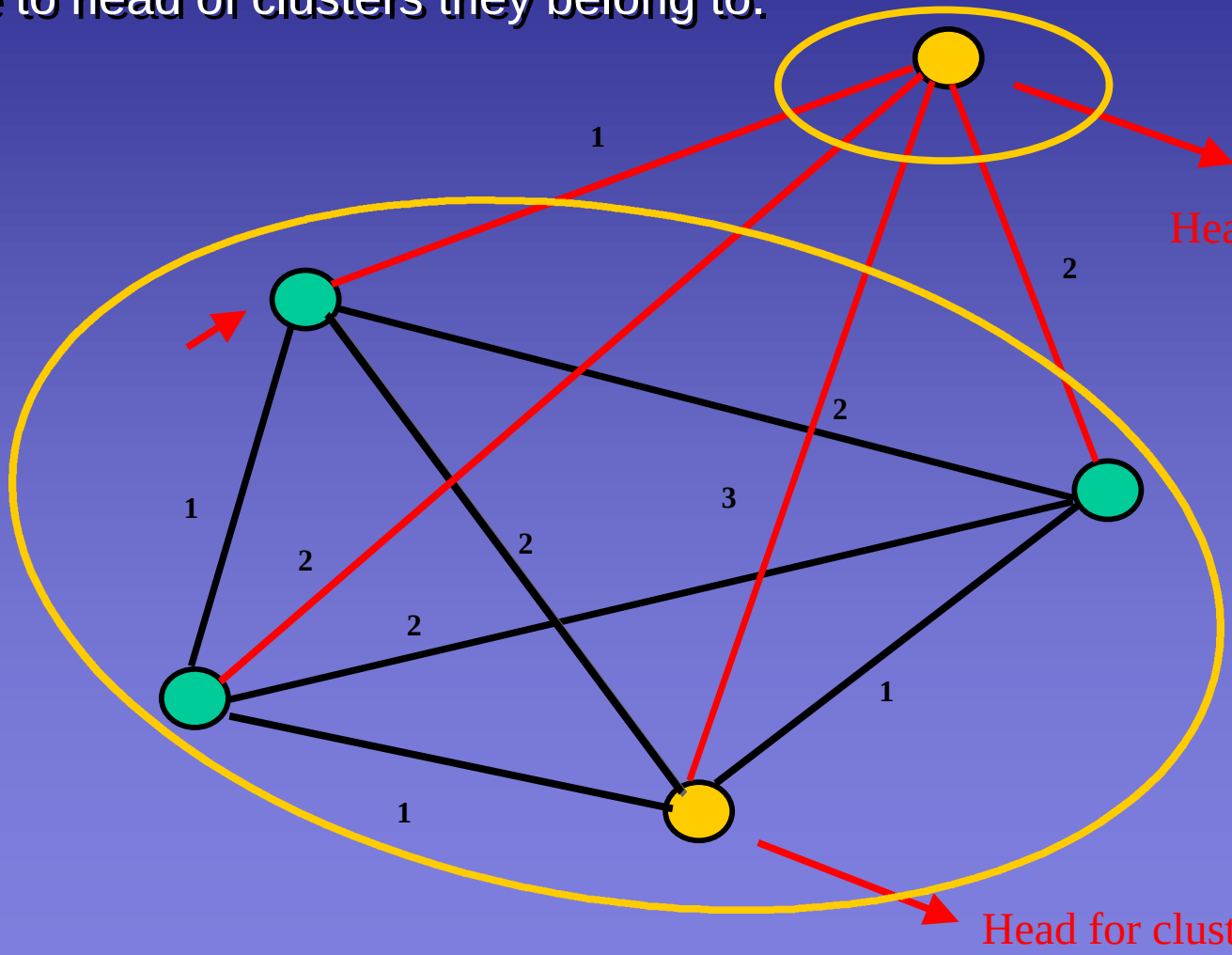
- Move the nodes inside the clusters connected to head of new cluster with lower weight compared to weight of edges connecting those to head of other clusters.



ER UCLA

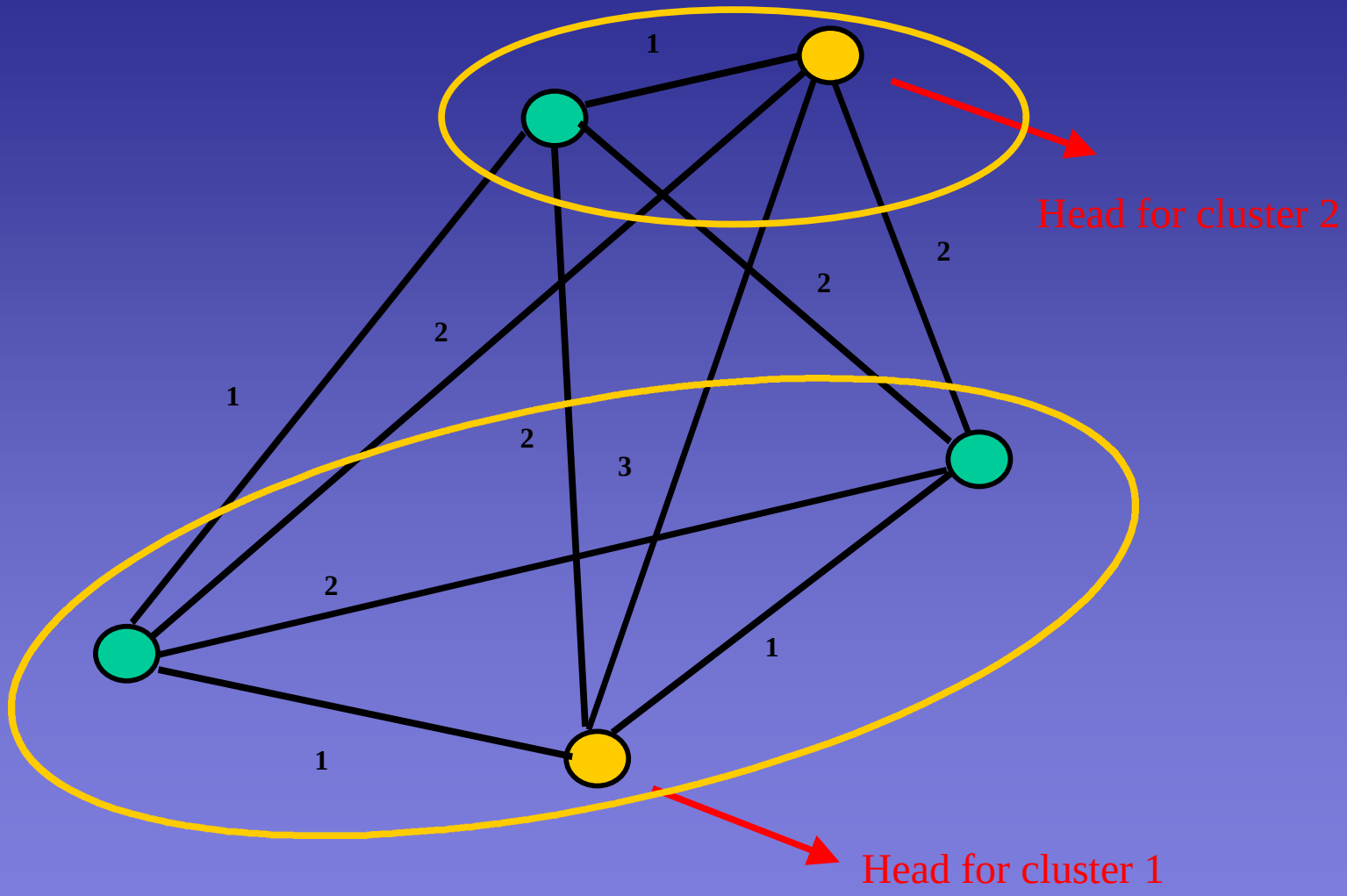
Example: Clustering Algorithm

- Move the nodes inside the clusters connected to head of new cluster with lower weight compared to weight of edges connecting those to head of clusters they belong to.



ER UCLA

Example: Clustering Algorithm



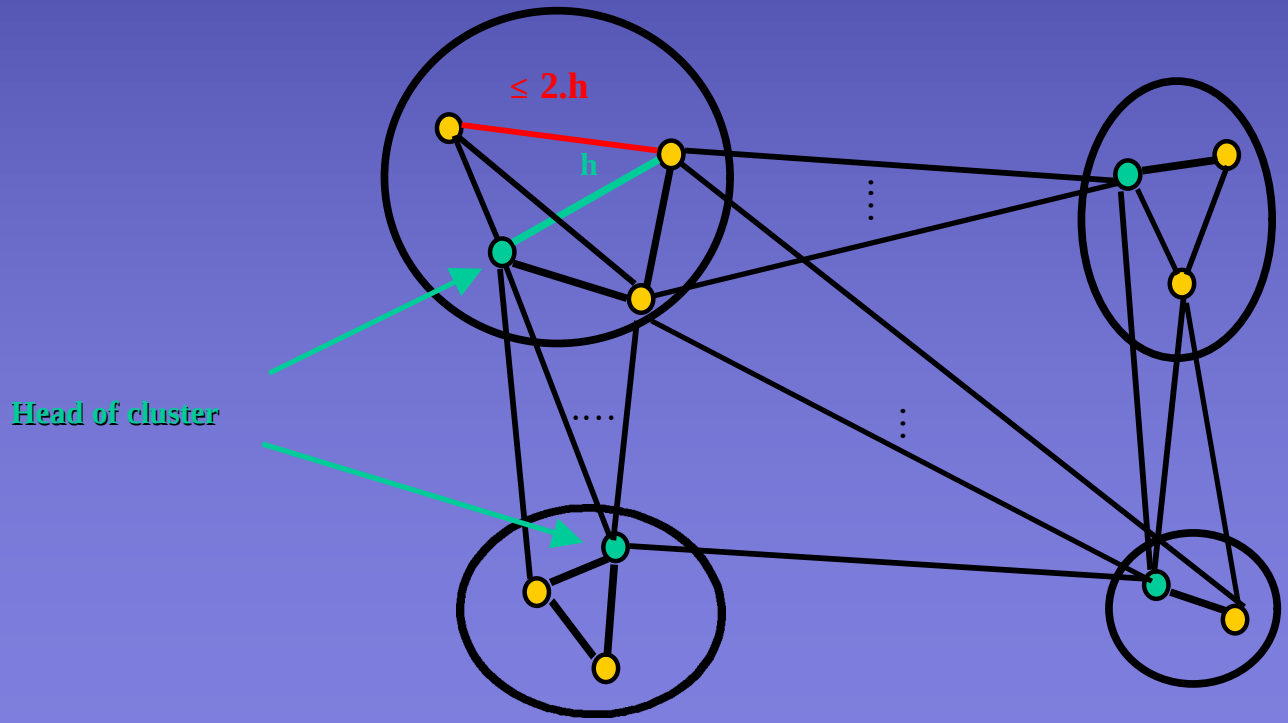
ER UCLA

Example: Clustering Algorithm

- It has been proven:
  - Proposed algorithm gives a solution within 2 times the optimal solution ($\alpha = 2$).
  - Computed approximation bound is the best possible for such problem (unless P=NP).
  - This implies that problem of approximation within $2-\varepsilon$ is NP-Complete.

Analysis of 2-approximation clustering algorithm

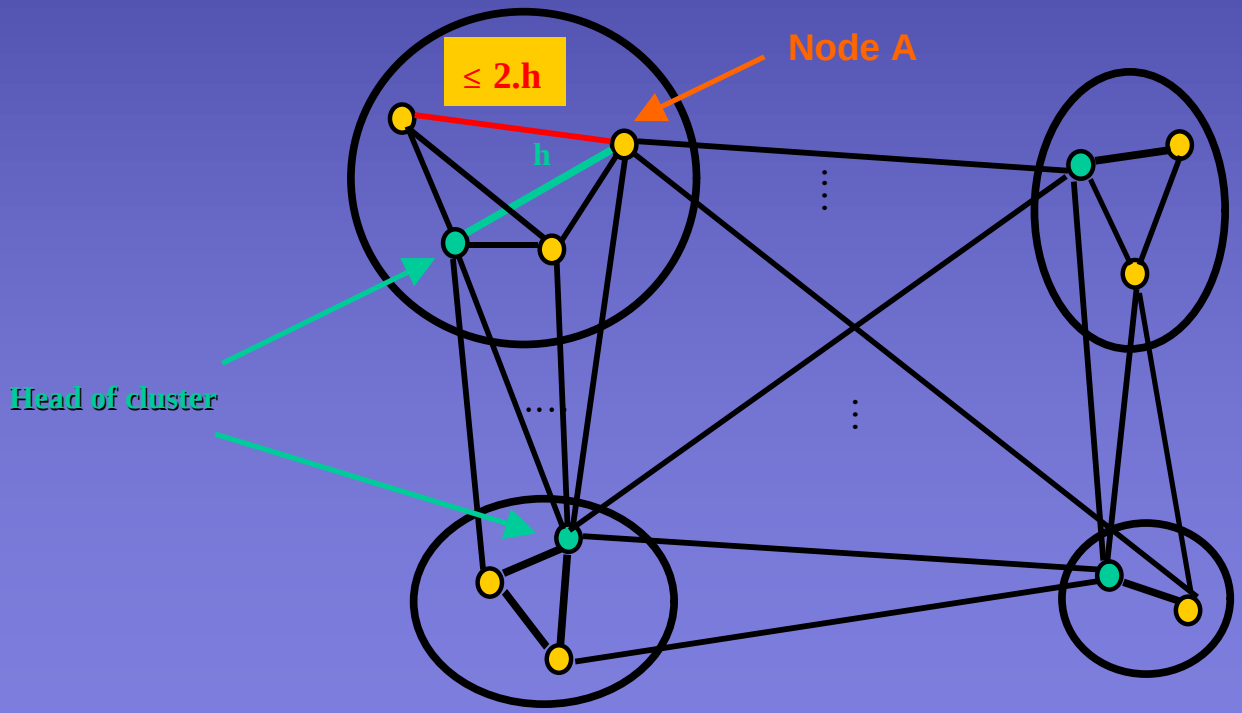
- h is defined as the maximum weight of edge inside clusters connecting to head of cluster.
- According to triangle inequality no edge in clusters has greater weight than 2 x h (G is a complete graph).



ER UCLA

Analysis of 2-approximation clustering algorithm

- Node A is connected to head of cluster with weight h.
- Since Node A has not been chosen to be head of other clusters (before or after cluster 1 was constructed), it implies that the weight of edges connecting Node A to heads of other clusters are at least h.



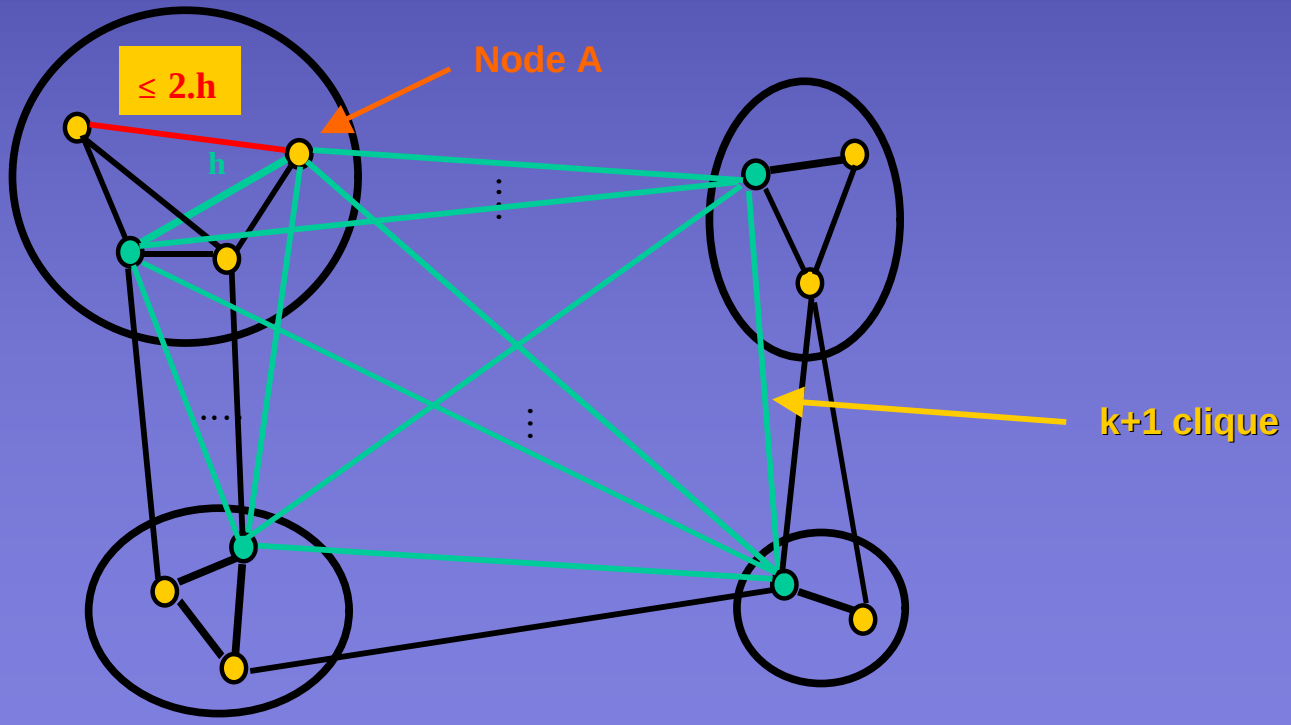
ER UCLA

Analysis of 2-approximation clustering algorithm

- Node A and heads of clusters (k nodes) generate a k+1 clique.
- Weights of edges in clique is at least h.
- If there is a (k+1)-clique with edges having greater weight than h, at least one edge of this clique will be inside one of the k clusters.

(k+1)-clique of weight h observed in G

→ $f^*(G) \geq h$

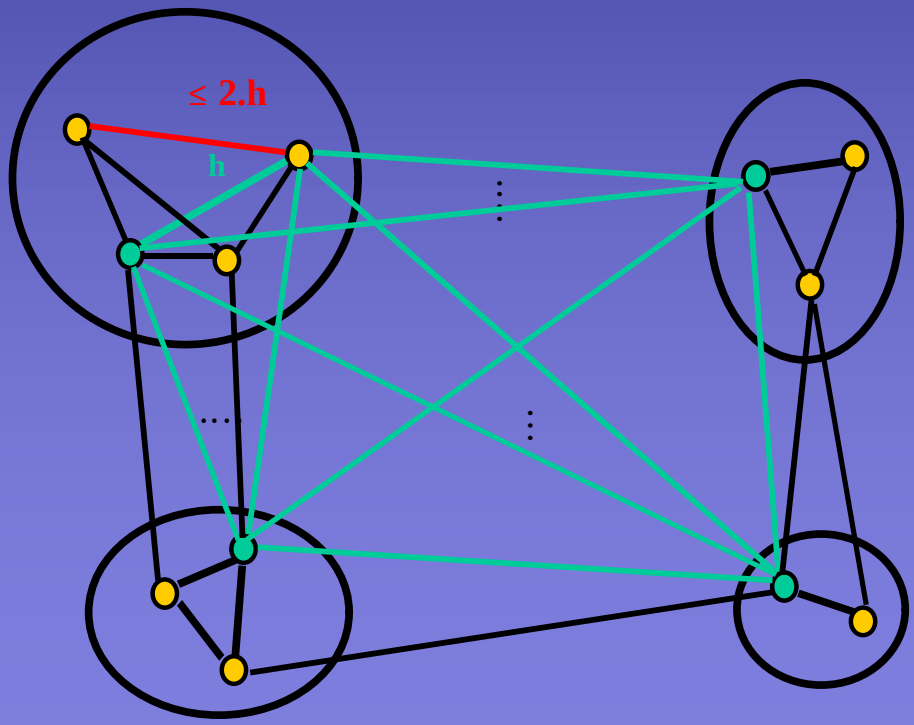


ER UCLA

Analysis of 2-approximation clustering algorithm

- Maximum weight of an edge inside any cluster is great than h.
- In solution obtain by approximation algorithm, no edge with weight greater than 2 x h is inside any cluster.

$$f(A,G) \leq 2 \cdot f^{*}(G)$$



ER UCLA

Conclusions

- Important to analyze algorithms
- Better to start with a simple algorithm, analyze it, add more local optimization methods, and improve the performance
- Let computational effort guide us towards the correct road to develop better algorithms for hard problems.

Probabilistic Algorithms

Probabilistic Algorithms

- Definition – an algorithm that makes random choices during execution
- Examples
  - Simulated Annealing
  - Genetic Algorithms
  - Karger's Contraction Algorithm for Clustering
- Use probablistic analysis to give:
  - Expected runtime
  - Bound on solution quality

Probabilistic algorithms often run faster, are easier to implement and describe than comparable deterministic algorithms

Contraction Algorithm for Clustering/partitioning

Input: directed acyclic graph $G = (V, E)$, number of partitions k

Output: k sets of vertices

begin

while |V| > k

do choose and edge $e(u,v)$ at random

contract both u and v

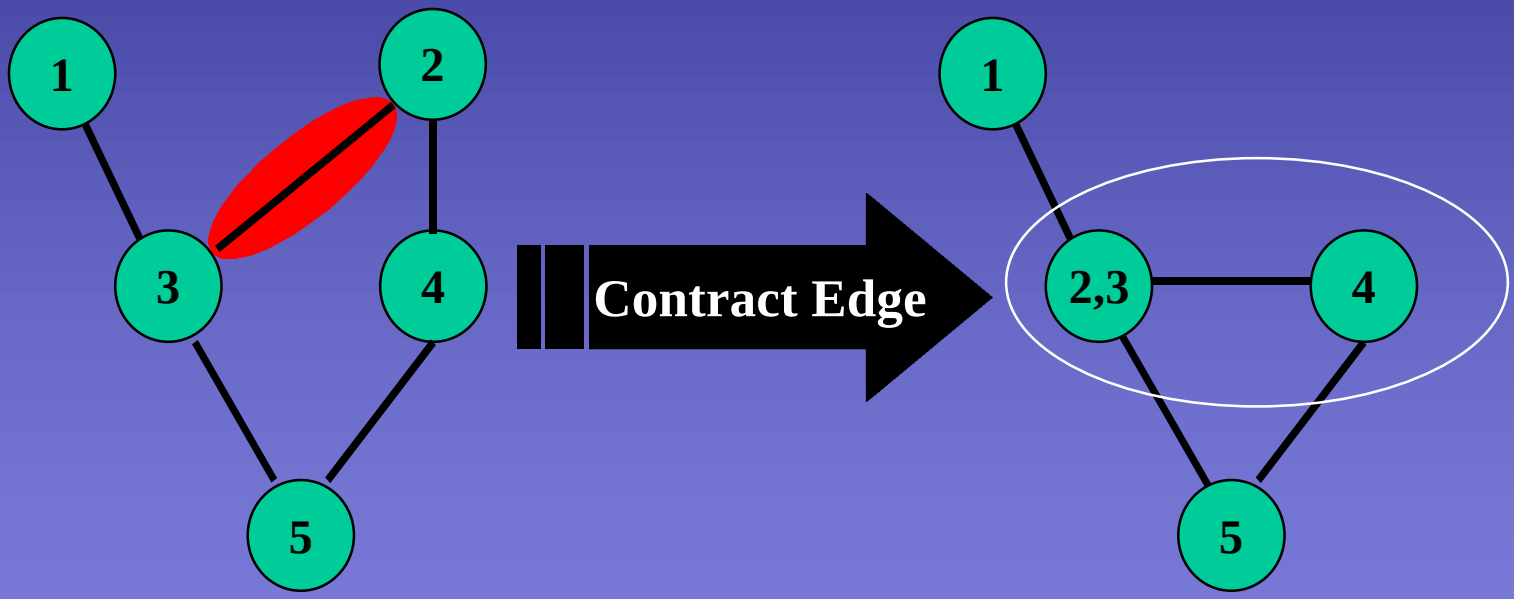
end

Simple to describe with good runtime and bound on solution quality

ER UCLA

Example

- Edge contraction
  - Remove vertices connected to edge
  - Replace with one vertex and maintain connectivity



"Simple" Algorithm with Analysis yields powerful results

ER UCLA

Solution Quality Properties

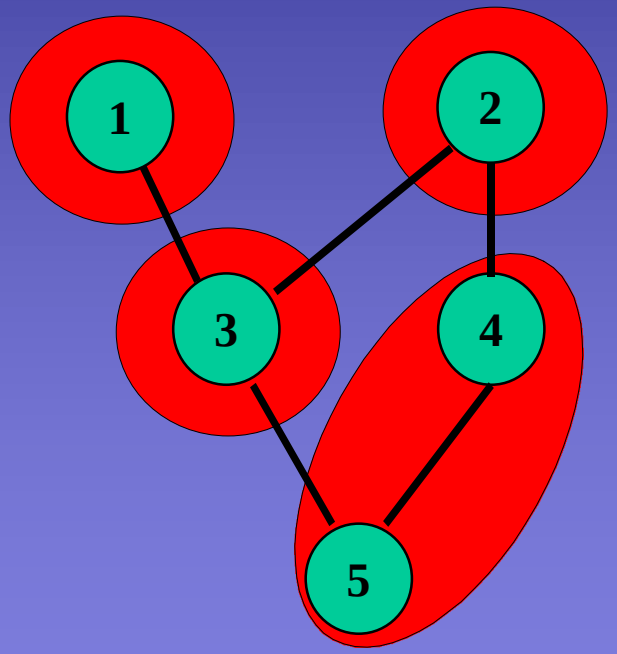
- *Using probabilistic analysis we can get bounds on solution quality*
- ***Theorem:*** *Stopping the Contraction Algorithm when r vertices remain yields a minimum r-way cut with probability at least*

$$r\binom{n}{r-1}^{-1}\binom{n-1}{r-1}^{-1}$$

- ***Corollary:*** *If we perform $O(n^2 \log n)$ independent contractions to two vertices, we find a min-cut with high probability.*
- *You want it faster: combine this with FM*

Proof of Theorem

- Choose *r - 1* vertices at random, each of these vertices is in its own cluster (*r* = 4)
- The remaining vertices are in the final cluster



- *f* is the number of edges cut (*f = 4*)
- *m* is the number of graph edges (*m = 5*)

$$E[f] = \left[1 - \left(1 - \frac{r-1}{n}\right)\left(1 - \frac{r-1}{n-1}\right)\right] m$$

Probability that a single edge is cut

ER UCLA

Proof of Theorem (cont)

$$E[f] = \left[1 - \left(1 - \frac{r-1}{n}\right)\left(1 - \frac{r-1}{n-1}\right)\right] m$$

- *f* is no less than the value of the minimum *r*-cut, E[*f*] is the value of the minimal cut
- The probability that a particular minimum r-cut survives the reductions process until there are r vertices remaining is at least

$$\prod_{u=r+1}^{n}\left(1 - \frac{r-1}{u}\right)\left(1 - \frac{r-1}{u-1}\right) = \prod_{u=r+1}^{n}\left(1 - \frac{r-1}{u}\right)\prod_{u=r+1}^{n}\left(1 - \frac{r-1}{u-1}\right)$$

$$= r\binom{n}{r-1}^{-1}\binom{n-1}{r-1}^{-1}$$

Insights into Clustering Problem

- Probability theory also gives tremendous insight into the clustering problem

- **Corollary:** *The number of minimum r-cuts of a graph is no more than* $O(n^{2(r-1)})$.

- **Corollary:** *The number of r-cuts within a factor of k of the optimum is* $O(n^{2k(r-1)})$.

"Simple" probabilistic algorithms give quality and runtime bounds and insight into the problem

Don’t be afraid to try wild ideas

- Crossing-based placement
- Embedding into intermediate graphs (netlist to mesh)

Final Comments

- Algorithm innovation is needed in all aspect of PD
- More so, as problems are getting more complex
- We need to look at
  - new paradigms,
  - novel algorithms,
  - concepts that may not go into a CAD tool (right away)
- Who is responsible for doing all these?
  - Industry
  - Academia