Approximation Techniques for Coloring Problems

Magnús M. Halldórsson
University of Iceland

Focus

- How well can we color graphs?
- How can we color graphs reasonably well?

- What are the techniques that we know?

Philosophy / Motivation

- Illustrate the wide span of coloring questions
- Introduce results ready for improvement
  - Some classical results
  - Some recent work

Topics

1. (Ordinary) Graph Coloring
2. Color Saving & k-Set Cover
3. IS in Hypergraphs
4. Scheduling with Conflicts
5. Coloring Bounded-degree Graphs

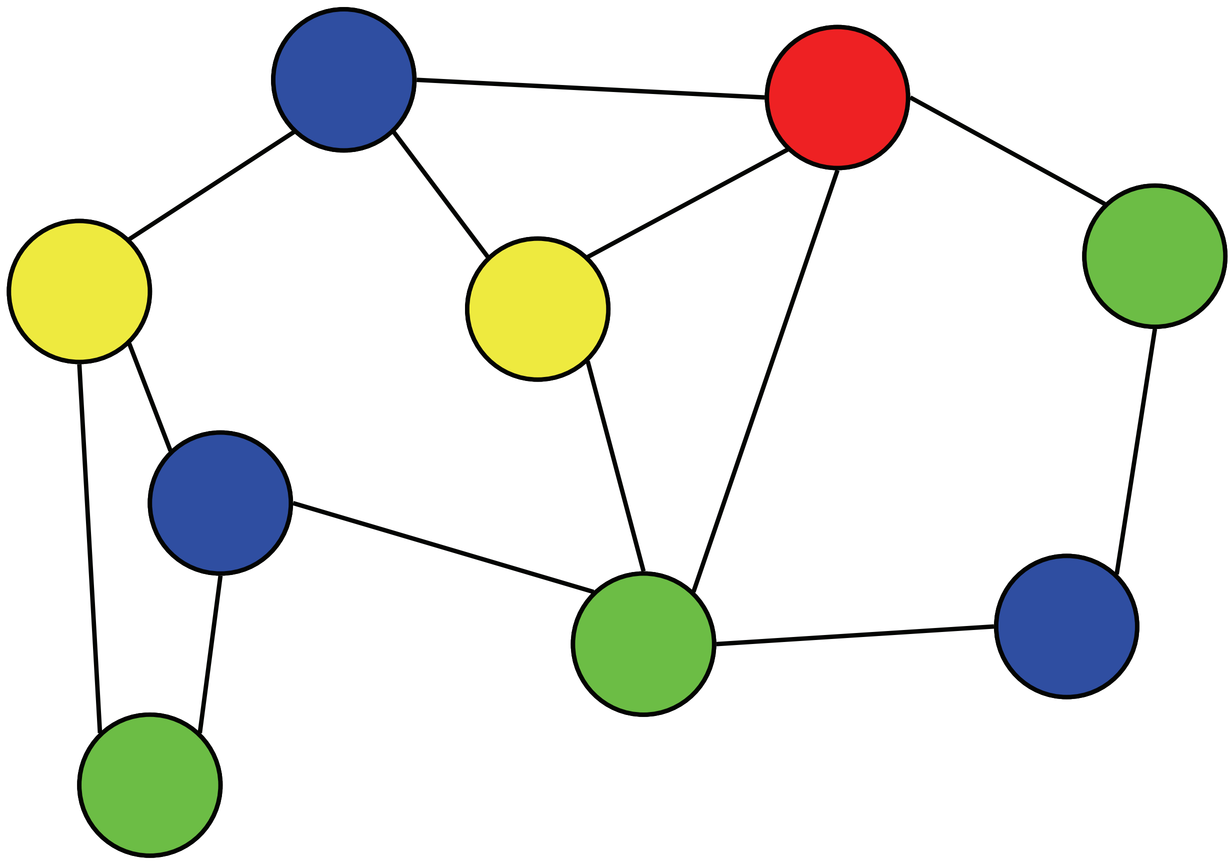
Techniques

- Greedy algorithms
- Local search & Semi-local search
- Semi-definite programming
- Partitioning & Randomized partitioning
- Other

Part I:
Classical Graph Coloring

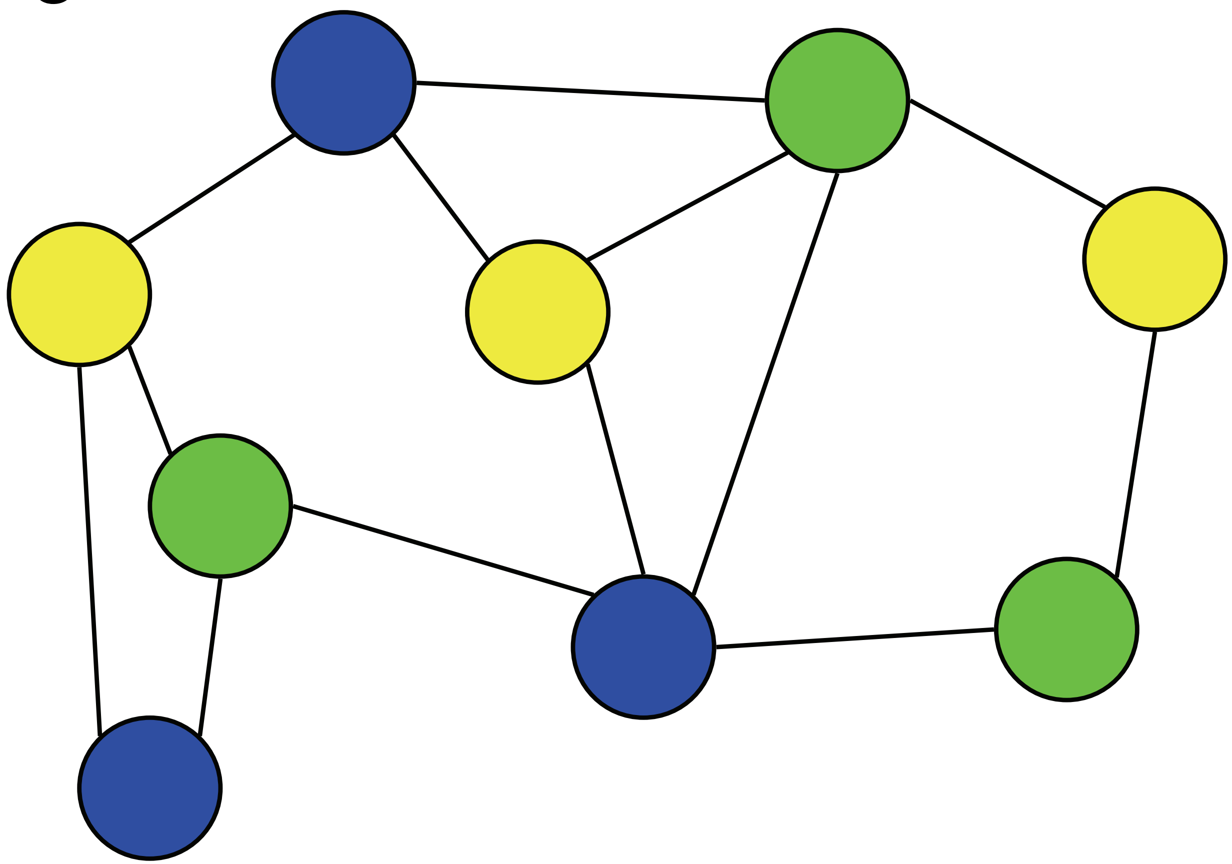
A blast from the past

Graph Coloring



Chromatic Number

$\chi(G) = 3$



Notation

- $\chi(G)$: **Chromatic number**, minimum number of colors needed to color graph G
- $\Delta(G)$: **Maximum degree** of a vertex in G
- n : **Number of vertices** in G

Performance ratio

- We are interested in algorithms that have guaranteed good behavior.
- Want the number of colors used to be "close" to the optimum number.
- Performance ratio of algorithm A is the function
  - $\rho_A(n) = \max_{G \text{ on } n \text{ vertices}} \chi(G)/A(G)$

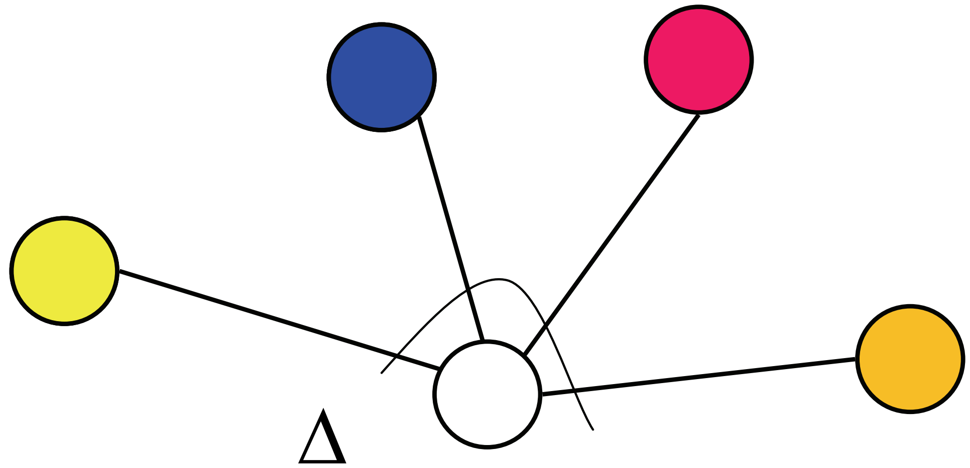
General graphs : Trivial bounds

- G is 1-colorable : Easy
- G is 2-colorable : Easy (linear time)
- So, we may assume $\chi(G) \geq 3$.
- But, $A(G) \leq n$, for **any** algorithm A

→ n/3-approximation, if we test for 2-colorability

Bounded-degree graphs: Trivial bound

- No algorithm needs to use more than $\Delta+1$ colors



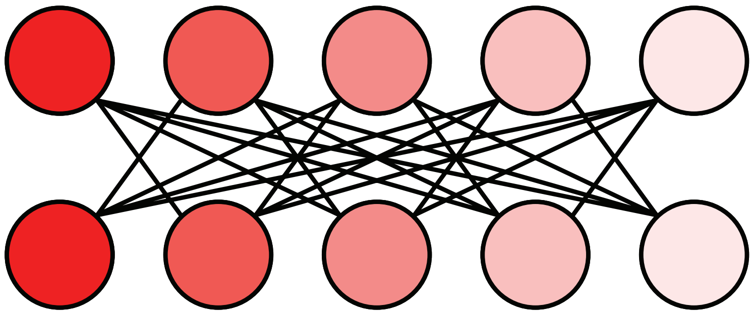
- $(\Delta+1)/3$ – performance ratio

First-Fit Algorithm

Online algorithm

First-Fit:
Process the vertices in arbitrary order:
  Assign each vertex the smallest possible color

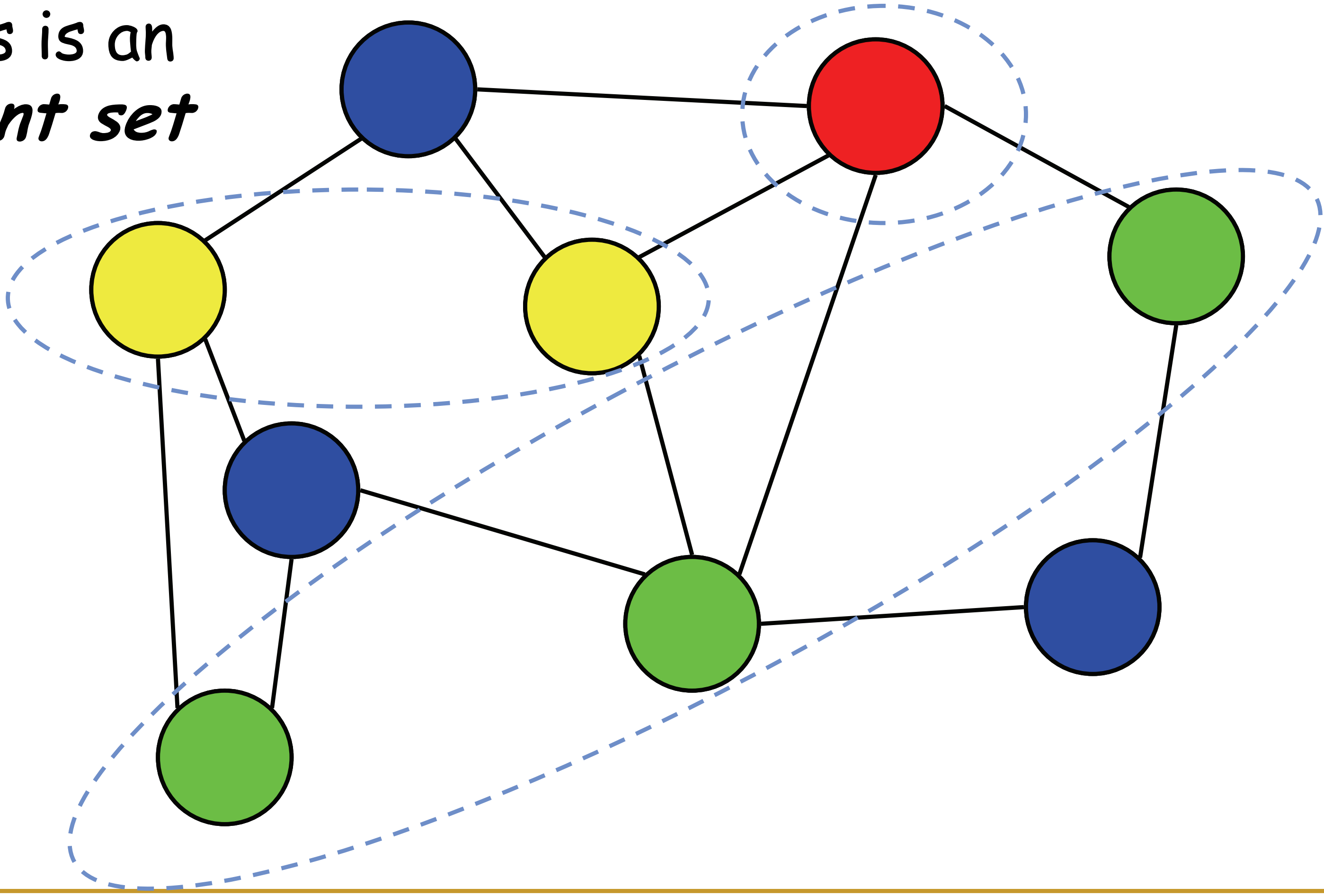
- Not good performance:



$$\rightarrow \rho_{FF} \geq \frac{n/2}{2} = n/4$$

Coloring & Independent Sets

Observation: Each color class is an *independent set*



Coloring by Finding Independent Sets

- A natural approach to coloring is to focus on finding large independent sets

Coloring-by-Excavations (schema):
While the graph is not empty **do**
    Find a large independent set
    Use a new color on those vertices

How Good is Excavating?

Remember, IS problem is also NP-hard

exact IS algorithm for excavating

log n –approximation for coloring

Not very good!

Excavation for Weaker Approximations

f(n)-approximation for IS, $f(n) = \Omega(\sqrt{n})$

O(f(n)) –approximation for coloring

Not bad !!

Proof of excavation lemma

- Count how many colors we need to halve the size of the graph
- χ f(n) colors needed to reduce vertices to n/2
  - There is a color of size at least $(n/2)/\chi$
  - IS algorithm finds one of size $(n/2)/(\chi\, f(n))$
- χ f(n/2) needed reduce vertices to n/4
- …..
- Total of $\chi\, (f(n) + f(n/2) + \ldots + f(\sqrt{n}))$ colors
- Geometric sequence, equals $O(\chi\, f(n))$,
  since $f(n) = \Omega(\sqrt{n})$

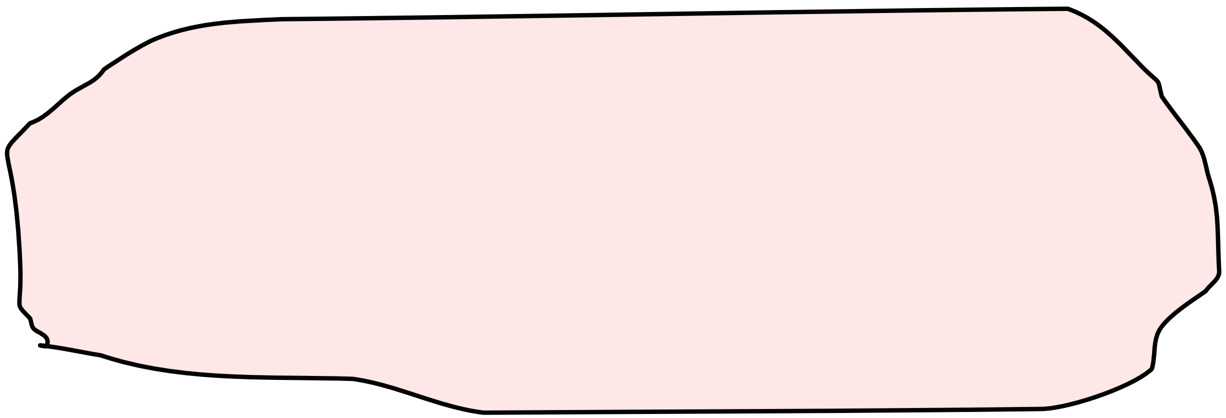
Modified goal

- Finding large independent sets in k-colorable graphs

Greedy IS (Johnson '74)

GreedyIS :
While the graph is not empty **do**
  Add a vertex of minimum degree to solution
  Remove its neighbors

Claim: There is always a vertex ***v*** with at least n/χ-1 *non-neighbors*

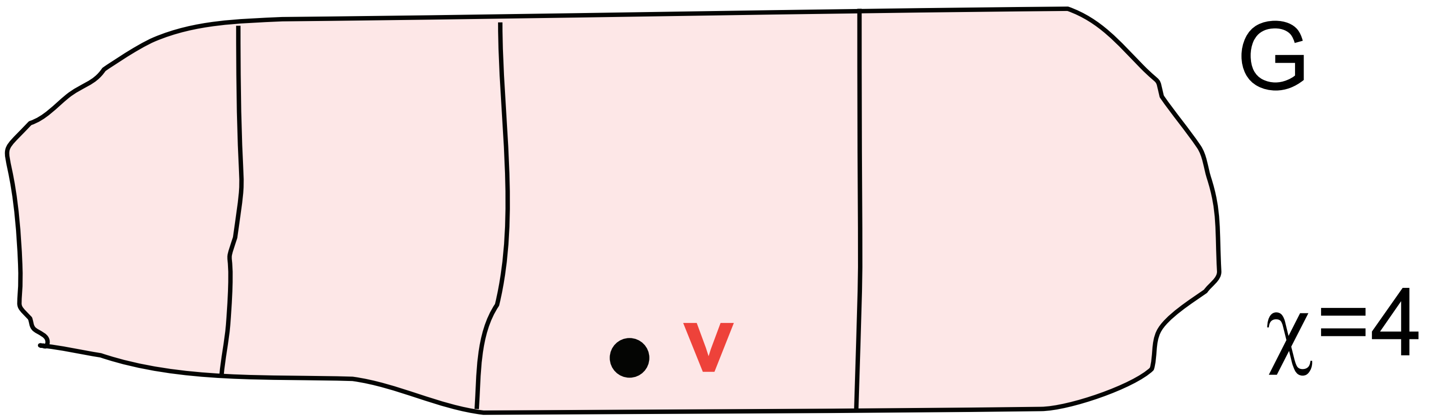


G

Greedy IS (Johnson '74)

GreedyIS :
While the graph is not empty **do**
  Add a vertex of minimum degree to solution
  Remove its neighbors

Claim: There is always a vertex ***v*** with at least n/χ-1 *non-neighbors*



Greedy IS (Johnson '74)

GreedyIS :
While the graph is not empty **do**
  Add a vertex of minimum degree to solution
  Remove its neighbors

Claim: There is always a vertex ***v*** with at least $n/\chi-1$ *non-neighbors*

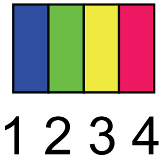
Claim: After t iterations, at least n/χ^t vertices remain

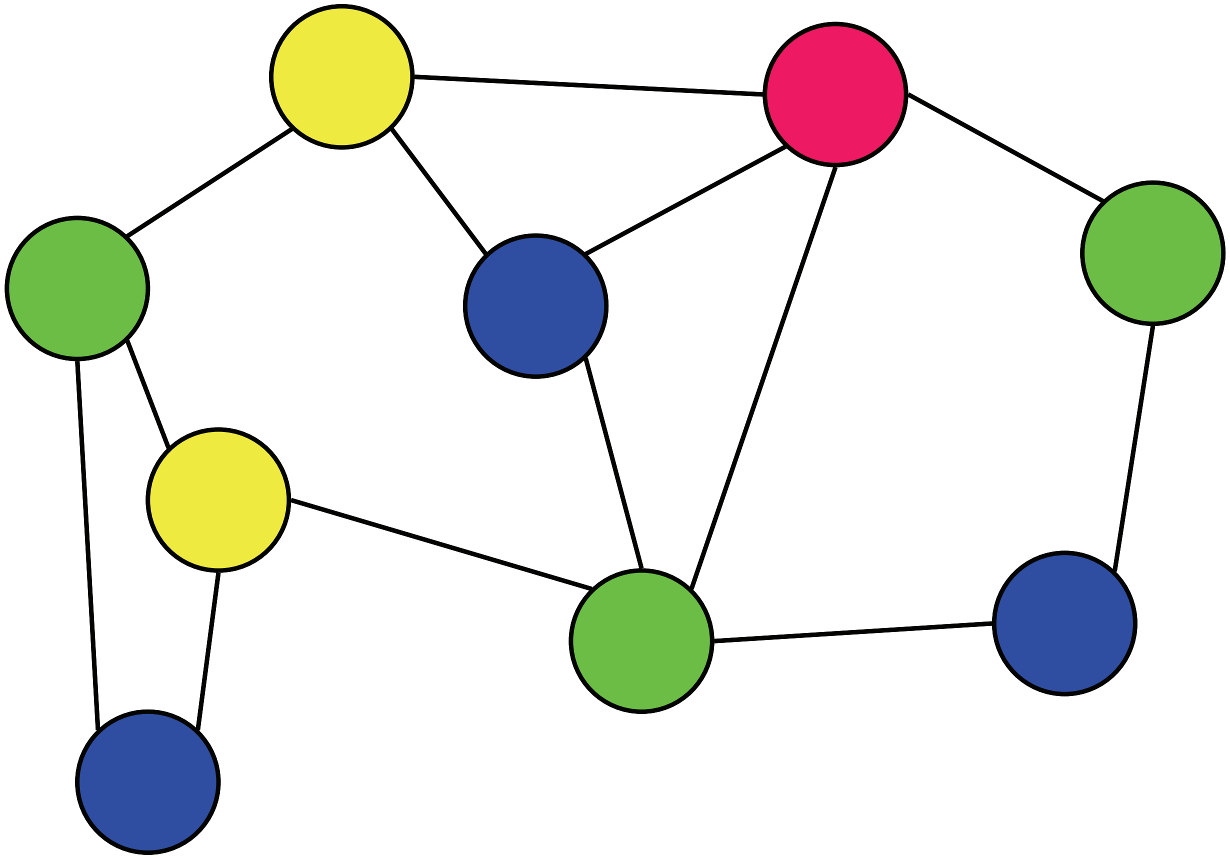
⇒ GreedyIS finds at $\mathbf{\log_\chi n}$ size IS

Performance ratio: $\chi/\ \mathbf{\log_\chi n \le n \lg\lg n/\lg n}$

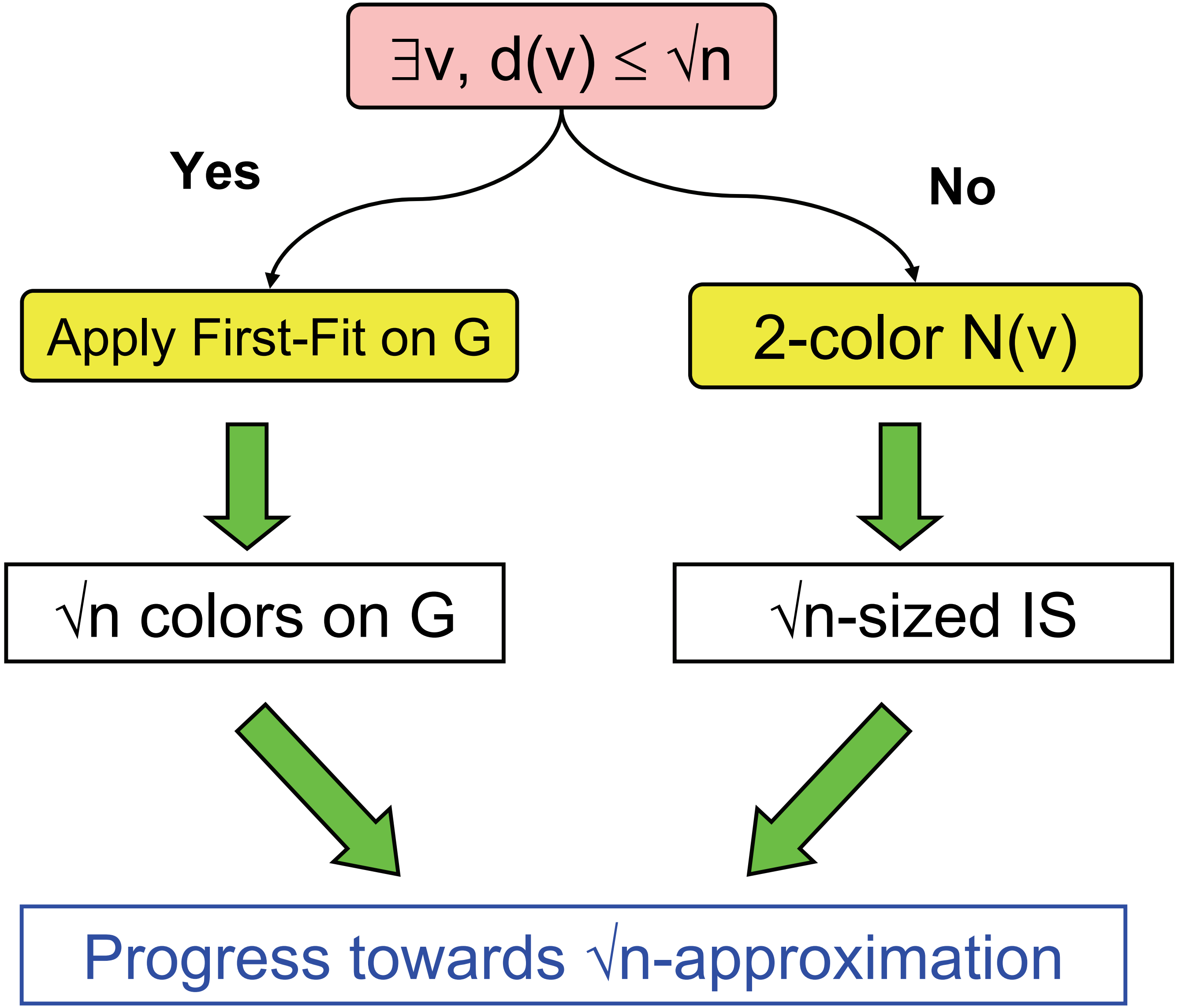
Equivalent Greedy Coloring Algorithm

Pick vertex with fewest uncolored neighbors and color it with smallest available color

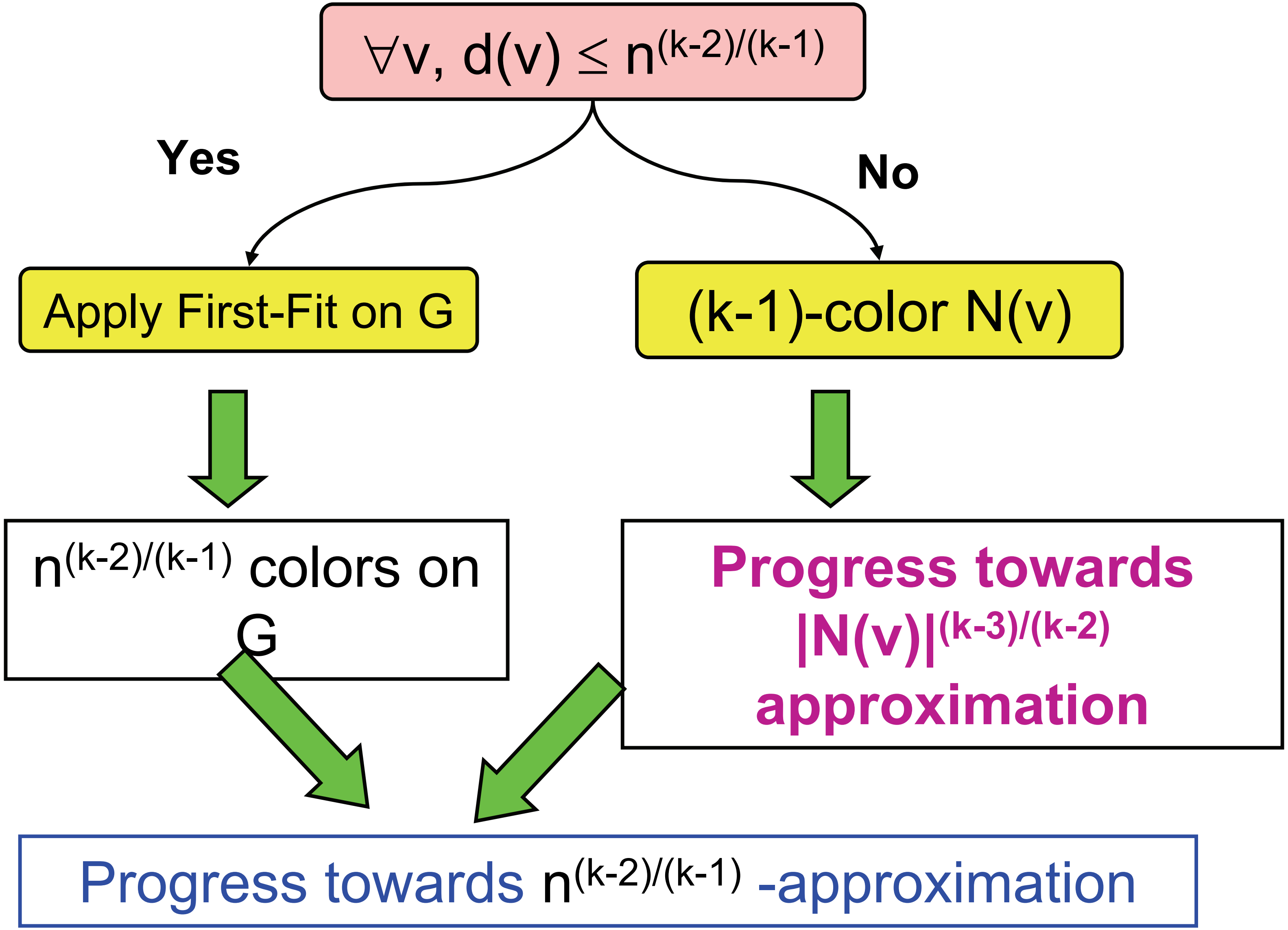




Wigderson: 3-colorable graphs



Wigderson: *k*-colorable graphs



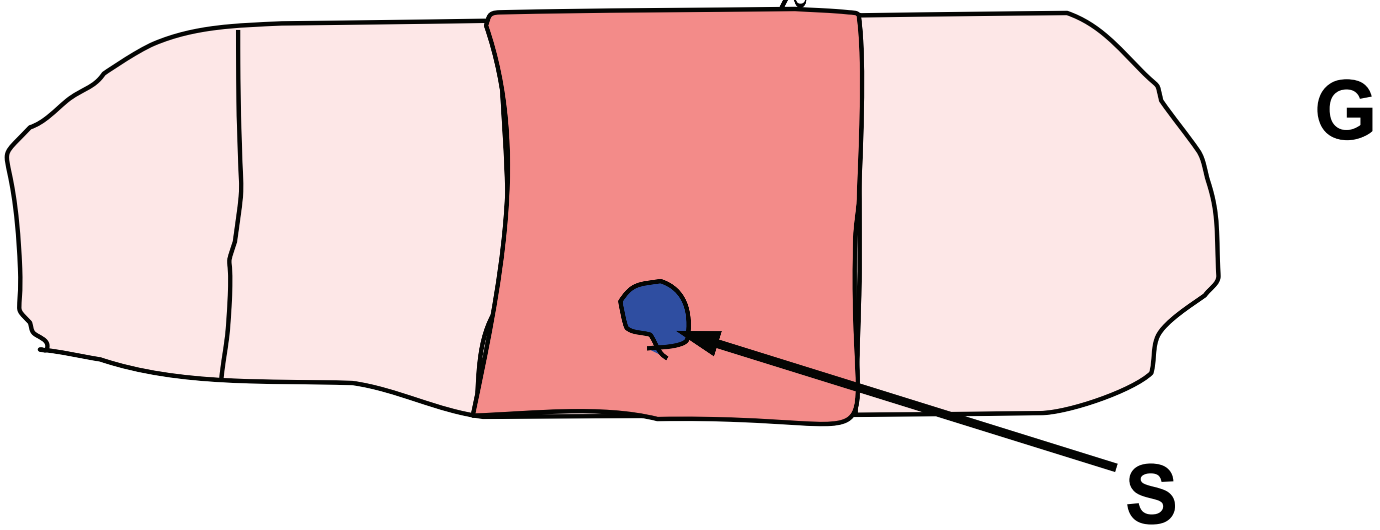
Berger-Rompel

Claim: There is always a vertex set **S** with

$N[S] \geq |V|/\chi$

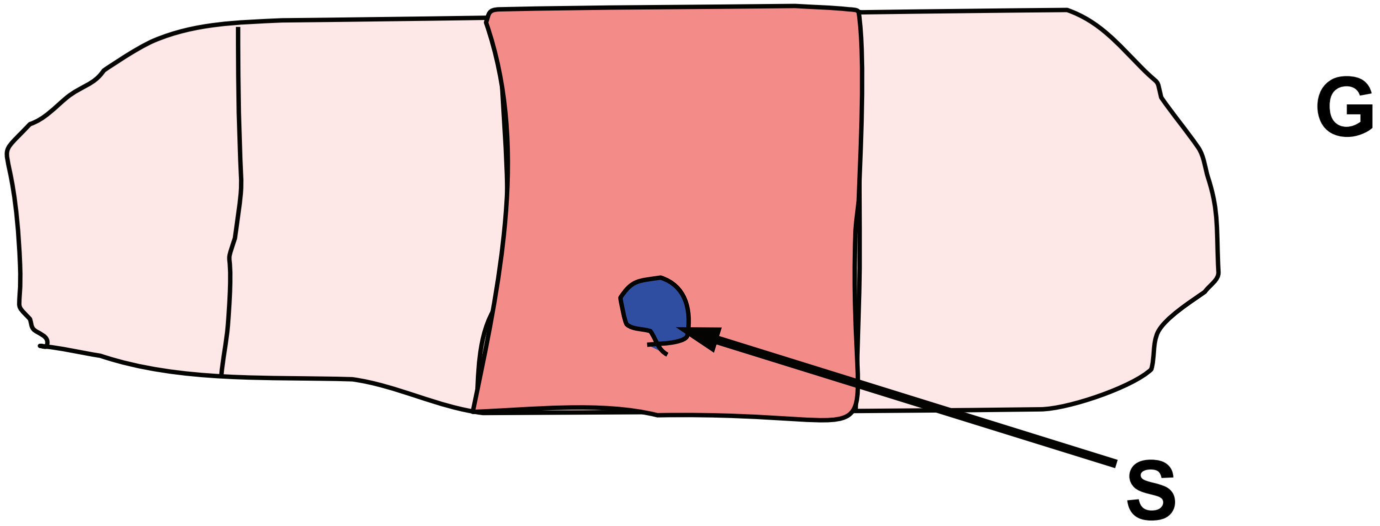
→ True for any set in **I**, the largest color class

→ Progress towards $|S| \log_\chi n$-approx.



Berger-Rompel

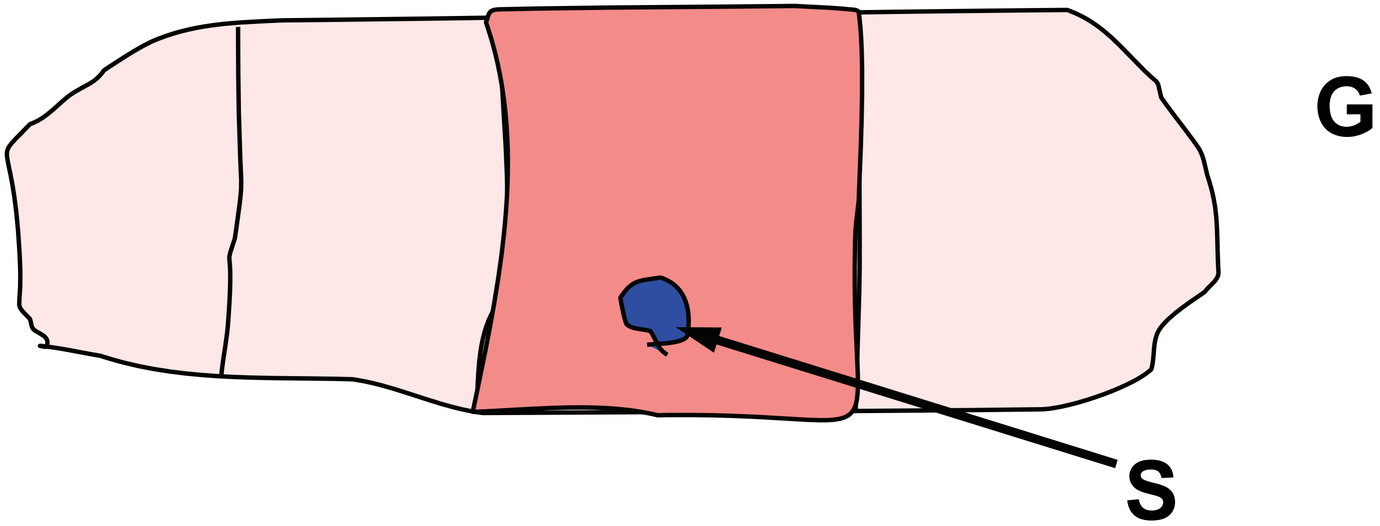
The number of k-sets **S** in **I** is at least $\binom{n/\chi}{k}$



Berger-Rompel

The probability that a random k-set is in I: $\dfrac{\binom{n/\chi}{k}}{\binom{n}{k}} \approx \left(\dfrac{1}{\chi}\right)^k$

This is 1/poly(n) when k = $\log_\chi$ n
In polynomial time, find a good $\log_\chi$ n-set.

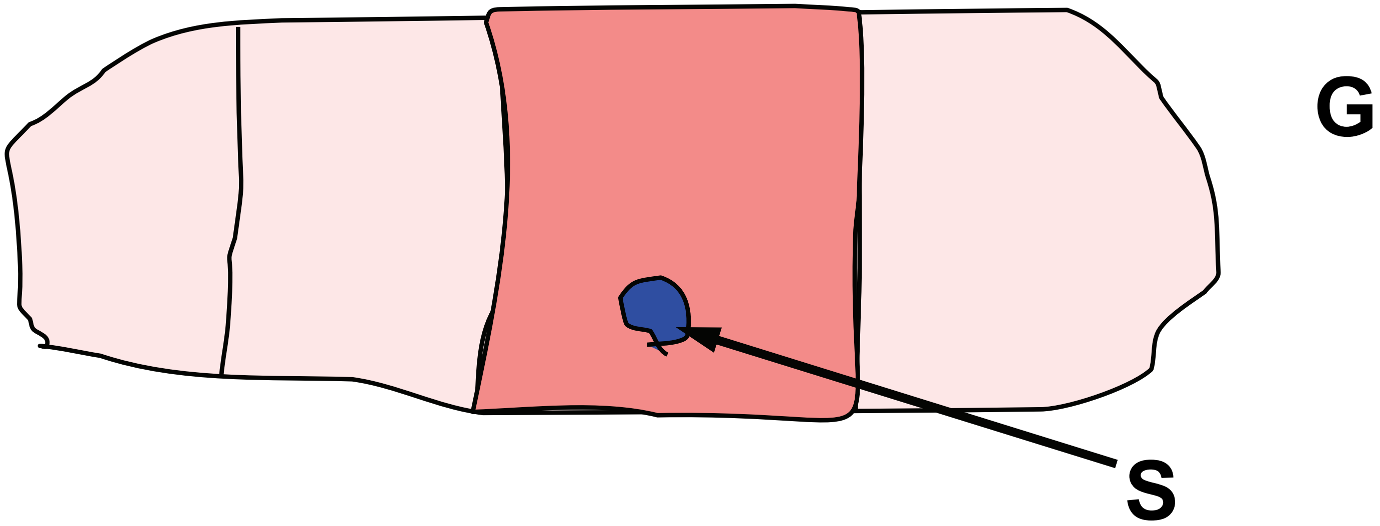


Berger-Rompel

In polynomial time, can find a **good** $\log_\chi n$-set S:

- S is independent
- S has at least n/χ non-neighbors

Recursively apply the search on G[V ¥ N(S)]

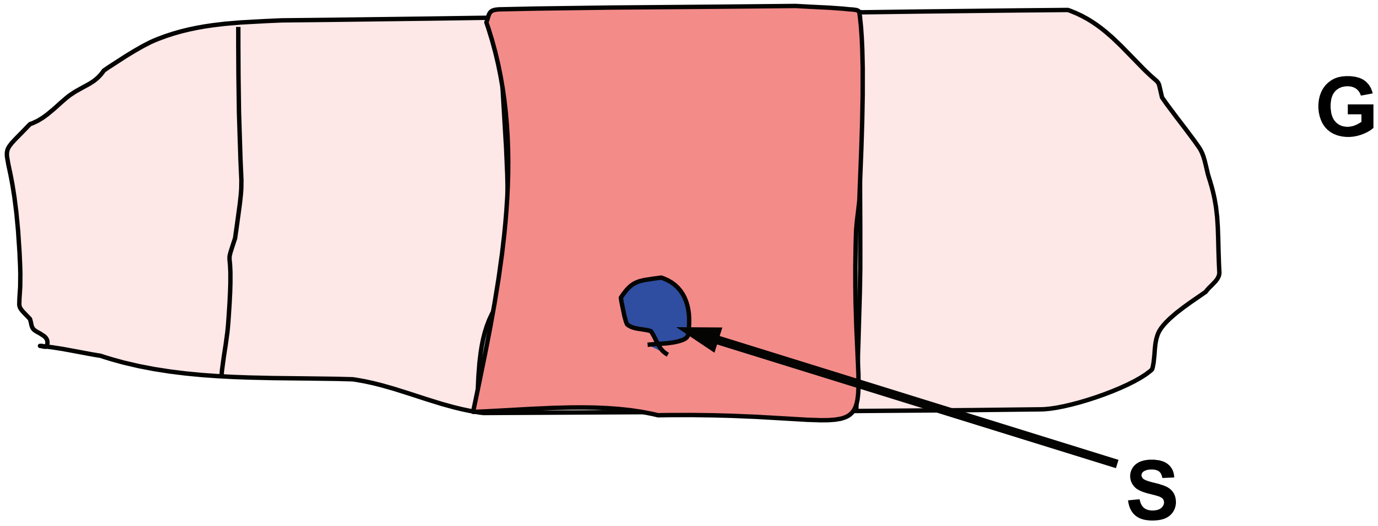


Berger-Rompel

Size of solution:

$f(n) = \log_\chi n + f(n/\chi)$ [Actually, $f(n/\chi - \log_\chi n)$]

Or, $f(n) = (\log_\chi n)^2/2$



Another view of Johnson's method

- We can find a vertex that *behaves like a vertex in a maximum IS*
  - Property: The vertex has many non-neighbors
- Because the graph is χ-colorable, we can apply this property recursively
  - Gives a $\mathbf{log}_{\chi}\, \mathbf{n}$ size solution

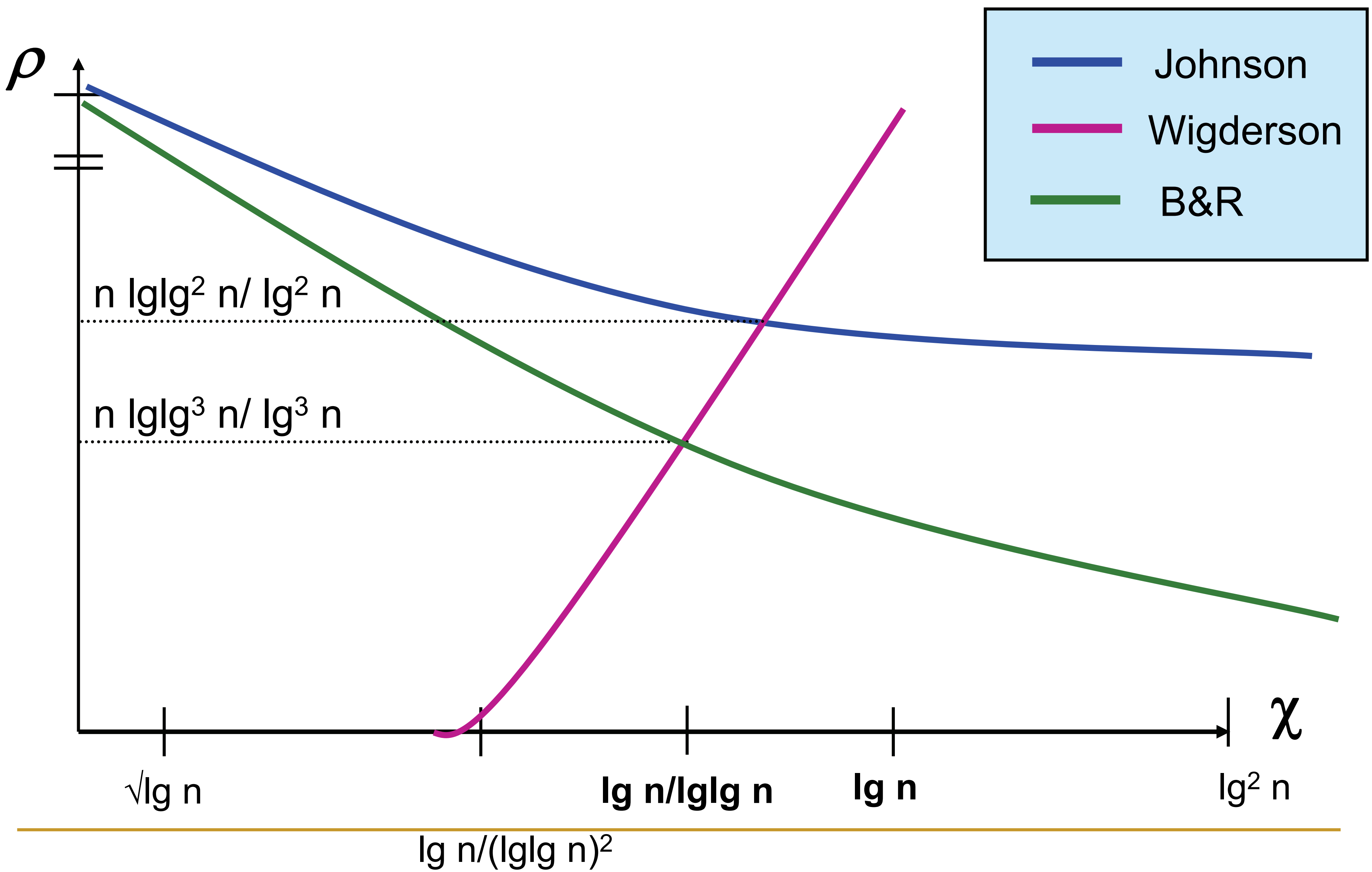
Another view of the B&R method

- We can find a $\log_\chi n$ -vertex set that *behaves like a subset a maximum IS*
  - Property: The set has many non-neighbors

At least n/χ

- Because the graph is χ-colorable, we can apply this property recursively
  - Can do $\log_\chi n$ rounds.
  - Gives a $(\log_\chi n)^2/2$ size IS

Performance ratios for Graph Coloring



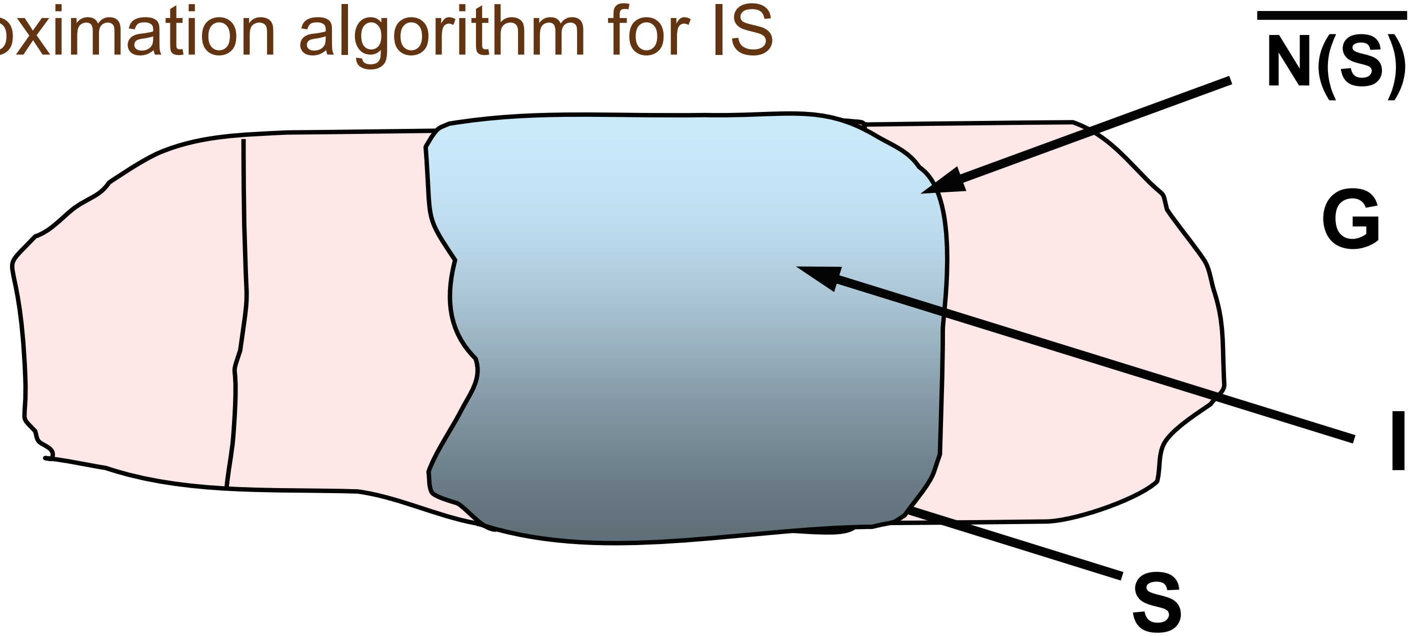
Overview of Upper Bounds

Johnson ´74	n/lg n
Wigderson ‘81	$n\,(\mathrm{lglg}\ n/\mathrm{lg}\ n)^2$
Berger&Rompel ’90	$n\,(\mathrm{lglg}\ n/\mathrm{lg}\ n)^3$
Halldórsson ‘91	$n\ \mathrm{lglg}^2\ n/\mathrm{lg}^3\ n$

- Best possible: n / polylog n ?

Improvement in [H '93], $\chi = \lg n / \lg\lg n$

- We can find a $\log_\chi n$ -vertex set that *behaves like a subset a maximum IS **I***
  - Property: The set has at least n/χ non-neighbors
  - If it has << n non-neighbors, then we can use an approximation algorithm for IS

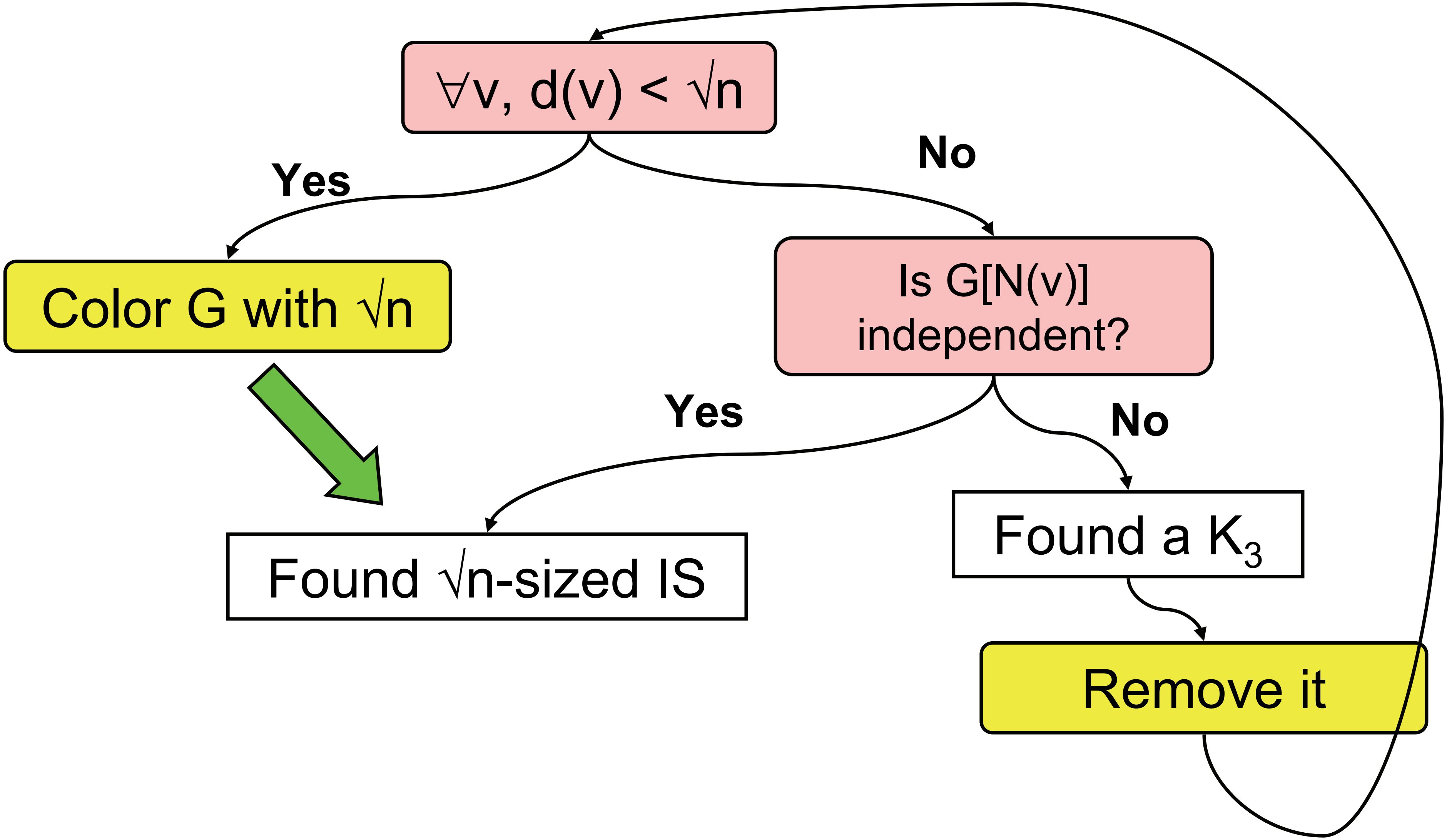


Improvement in [H '93], $\chi = \lg n / \lg\lg n$

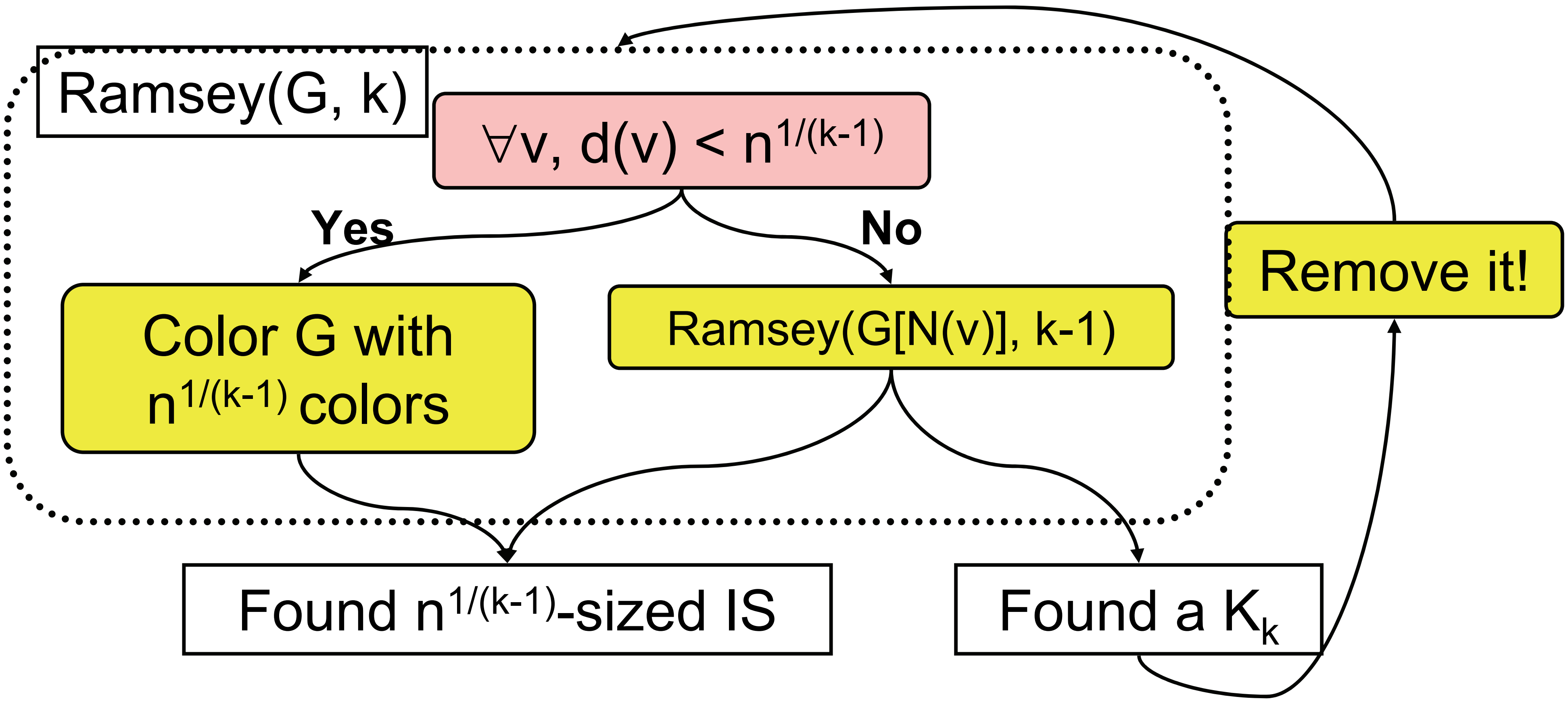
- We can find a **$\log_\chi n$** -vertex set that *behaves like a subset a maximum IS*
  - Property: The set has at least n/χ non-neighbors
  - If it has << n non-neighbors, then we can use an approximation algorithm for IS

- Because the graph is χ-colorable, we can apply this property recursively
  - Can do **log n** rounds
  - Gives a **$\log n\ (\log_\chi n)^2/2$** size IS

Clique Removal: Case $\alpha(G) > n/3$

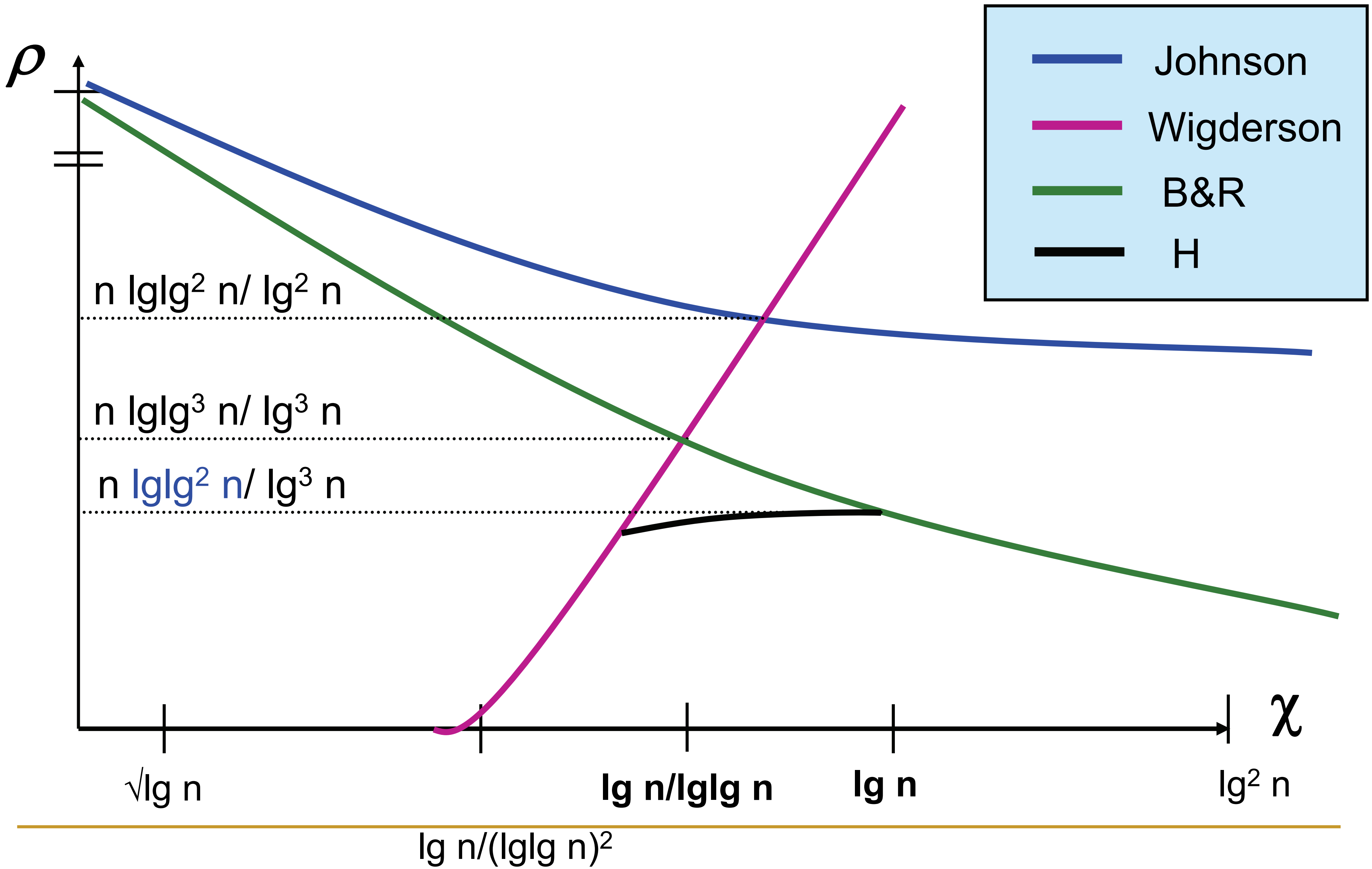


Clique Removal: Case $\alpha(G) > n/k$



Eventually finds a $O(kn^{1/(k-1)})$-sized IS $\Rightarrow$ $\rho \leq n/\log^2 n$

Performance ratios for Graph Coloring



Lower Bounds

- Sequence of impressive and often seminal work on interactive proof systems
- Current best lower bound: $n / 2^{(\log n)^{3/4-\varepsilon}}$
  - [Khot, Panduswami ’06; Zuckerman ’05]
  - Relates to approximability of LabelCover
- The most promising approach:
  - Lovasz’ theta number & SDP

Open questions

- Improve the long-standing upper bound
  - I have no special suggestions
  - Core issue: log n-colorable graphs
- Is the θ(n/polylog n) conjecture for the best possible performance ratio of Graph Coloring true?
  - True for some restricted variants, like online coloring

Part II: Color Saving

Coloring as a SetCover problem

Pushing the “local” in “local search”

[Duh, Furer, 1996]

Color Saving: Maximizing the number of "unused" colors

- If a coloring uses ALG colors, there are **n-ALG** "potentially unused" colors saved.
- Optimization identical to Graph Coloring
  - Differential approximation ratio
    $\rho \cong (n-\chi)/(n-ALG)$

Easy 2-approximation

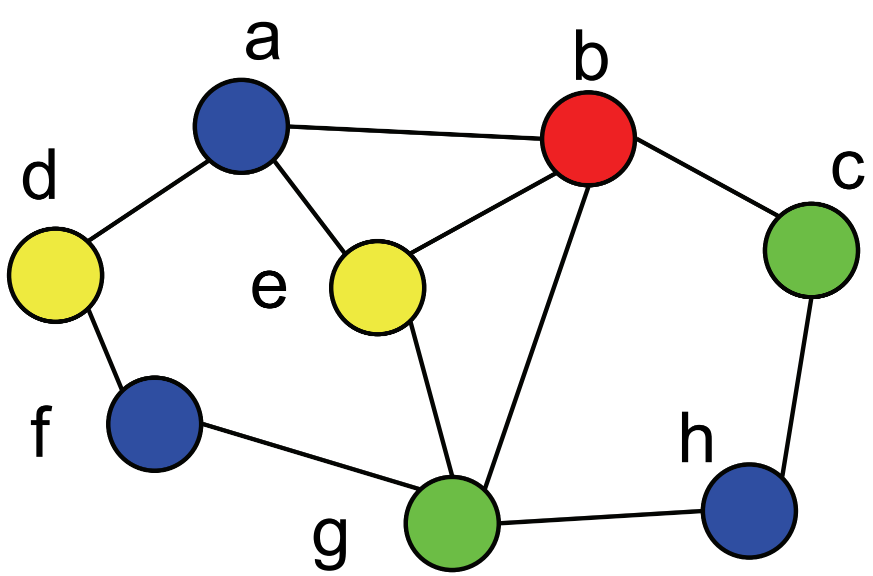
- Use at least 2 vertices per color, when possible
  - If A_1 = # color classes with a single vertex
  - $A_1 \leq \omega(G) \leq \chi(G)$
- Performance analysis
  - #colors used $\leq A_1 + (n - A_1)/2$

$$\rho \leq \frac{n-\chi}{n - \#\text{colors}} \leq \frac{n - A_1}{n - A_1 - (n - A_1)/2} = 2$$

Better Ratios for Color Saving

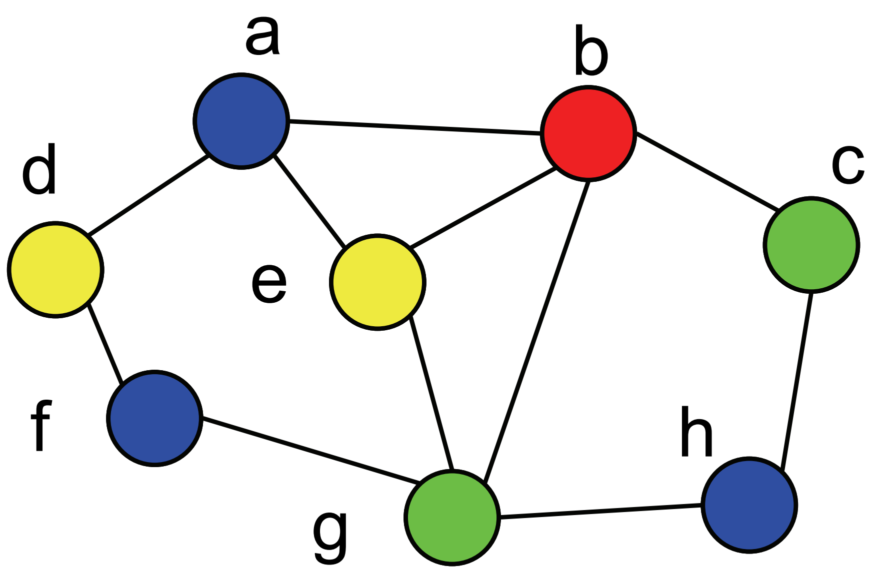
- We want 3-sets!
  - Suppose there are no 4-independent sets.
- Our problem now reduces to the following:
  - Find the smallest collection of independent sets of size 1, 2, 3, that covers all vertices.
  - Form a *set system* S over the ground set V:
    - S contains a set for each independent set in V
  - We seek a **minimum set cover** of S
    - k-Set Cover: Sets of size at most *k*.

Graph & System of 3-ISs



- V={a,b,c,d,e,f,g,h}
- S={acf,acg,afh, bdh,bfh,cde,cdg, cef,deh,efh}
  - & its subsets

Graph & System of 3-ISs



- V={a,b,c,d,e,f,g,h}
- S={acf,acg,afh,
  bdh,bfh,cde,cdg,
  cef,deh,efh}
  - & its subsets

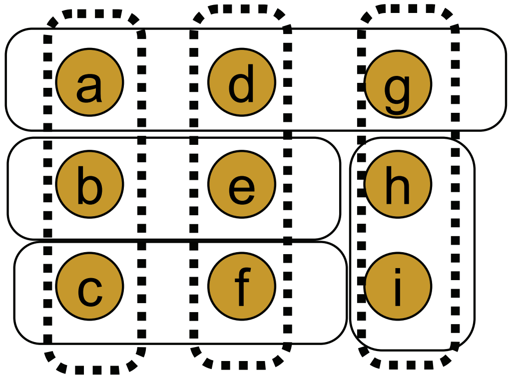
Disjoint Set Cover

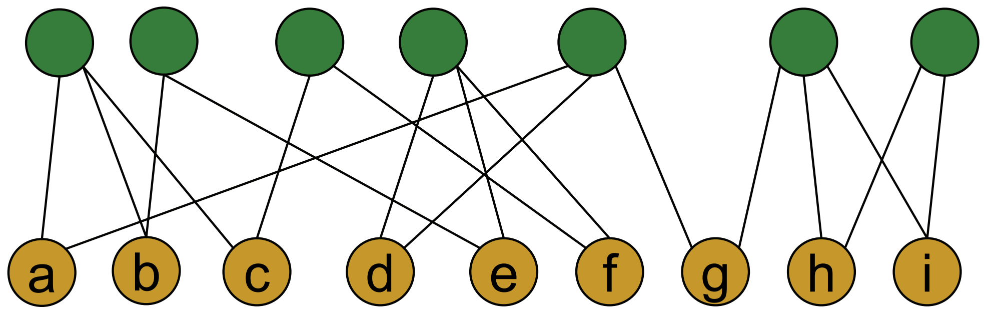
- It is convenient for us to assume that the set system is *monotone*:
  - If set S is in E, then S' is also in E, for $S' \subset S$.
  - E.g. if abc={a,b,c} $\in$ E, then a, b, c, ab, ac, bc $\in$ E
- Whenever one of the new set is used, we can replace it in the actual solution with a superset
- Increases instance by a factor at most 6

- Now, may assume the solution is ***disjoint***, i.e.
  a partition of S.

Minimum 3-Set Cover

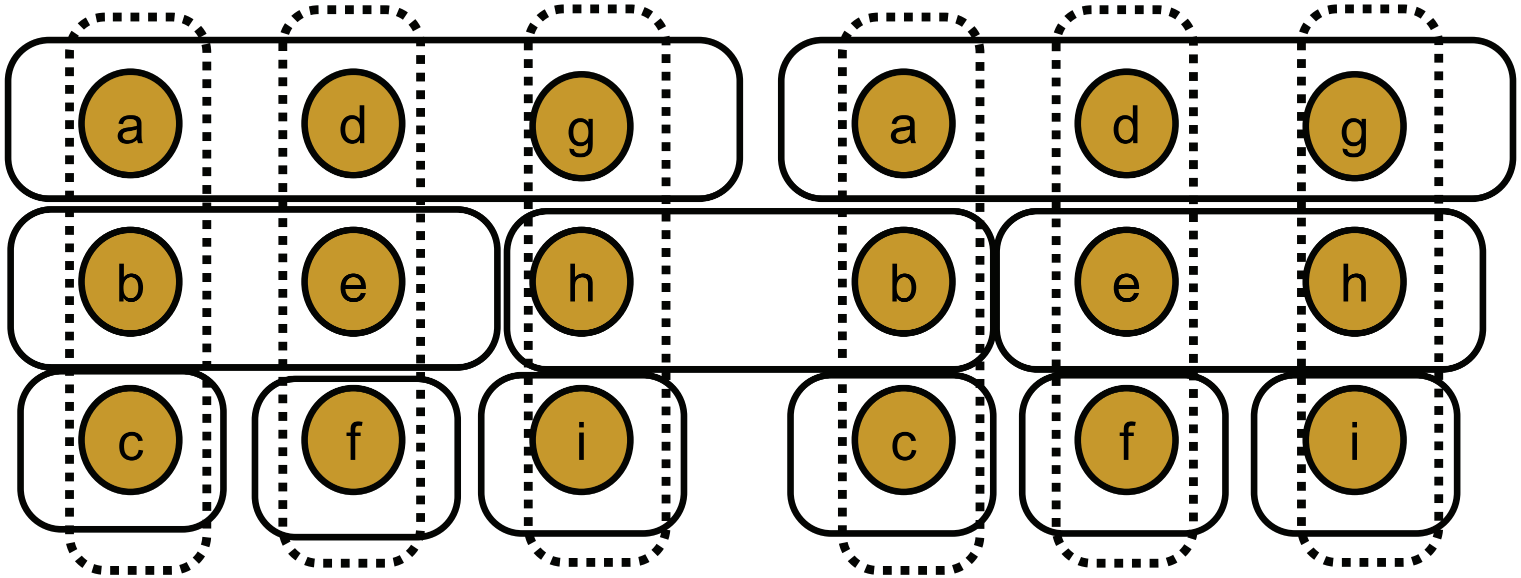
- Given:
  - Set S of base elements
  - Set E of subsets of S, **each of size at most 3**
- Example: E={ abc,def,ghi, adg, be, cf, hi}





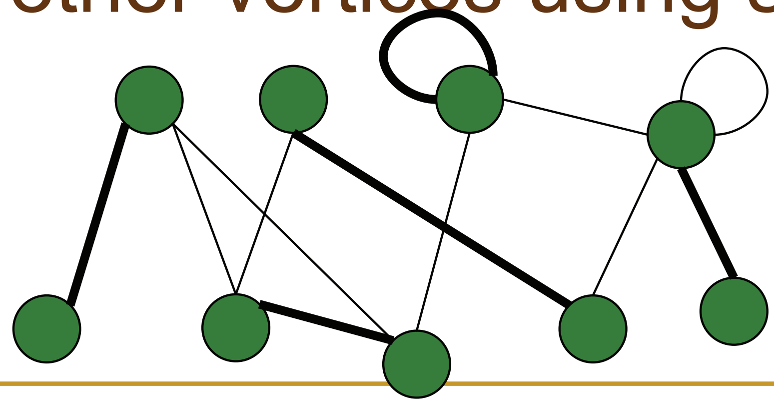
Greedy for 3-SC

- Greedy has approximation ratio
  $H_3 = 1 + ½ + 1/3 = 11/6$



2-Set Cover (= Edge Cover)

- If the sets are of size at most 2, then what we have is a *graph* with self-loops
  - A 2-set is an *edge,* a 1-set is a *self-loop*
- Solve by reduction to maximum matching:
  - Select edges of a maximum matching
  - Cover other vertices using self-loops or add'l edges



Using exact solution of 2-SC to help solving 3-SC

- Suppose we have fixed the 3-sets that we use in a solution.
- Then, we can find an *optimal* collection of 1-sets and 2-sets to cover the remaining elements.

Generic local improvement method

$S \leftarrow$ initial starting solution (obtained elsewhere)

while ($\exists$ small *improvement* I to S) do

  $S \leftarrow$ solution obtained by applying I to S

output S

A solution that has gone through local search is said to be **locally optimal** (with respect to the improvements applied)

Issues:

- What is an *improvement*? (Problem specific)
- How do we find the improvement? (Search)

Semi-local optimization for 3-SC

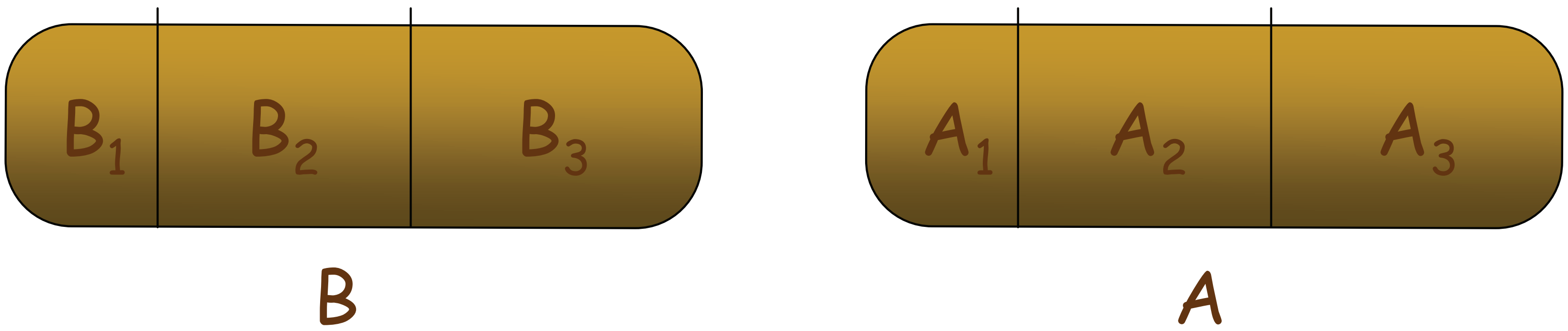
- Only the 3-sets in the solution stay fixed.
- A *(s,t)-change* consists of:
  - Adding up to *s* 3-sets
  - Removing up to *t* 3-sets
  - Finding an optimal 2-set cover of the remaining elts
- Objective function:
  - A) Minimize the number of sets in solution, or
  - B) Minimize the number of 3-sets in the solution
- **(s,t)-improvement**: An (s,t)-change with improved objective
  - Fewer A), or equal A) and fewer B)

Main result for 3-SC

- Theorem [Duh,Furer]:
  no (2,1)-semi-local improvement
  $\Rightarrow$
  4/3-approximation

Notation

- A : Algorithm's (2-opt) solution
- B : "Best" (optimal) solution
- A_i : "The collection of i-sets in A, for i=1,2,3
- B_i : "The collection of i-sets in B, for i=1,2,3
- $a_i = |A_i|$, $b_i = |B_i|$



Proof outline

- We will derive a few bounds on the sizes of the solution parts.
  - Obs 1: $a_1 + 2\,a_2 + 3\,a_3 = b_1 + 2\,b_2 + 3\,b_3 = |S|$
  - Lemma 2: $a_1 \leq b_1$
  - Lemma 3: $a_1 + a_2 \leq b_1 + b_2 + b_3$
- By adding the inequalities,

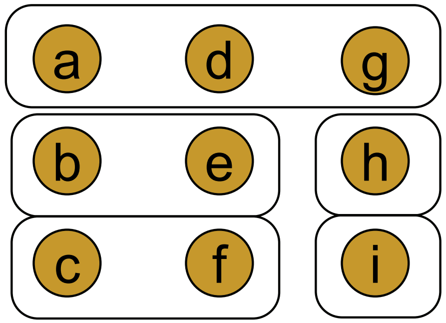
$$3\,a_1 + 3\,a_2 + 3\,a_3 \leq 3\,b_1 + 3\,b_2 + 4\,b_3$$

we get the theorem:

$$|A| \leq 4/3\,|B|$$

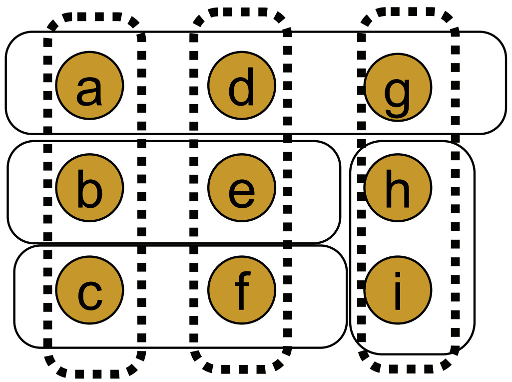
Observation 1:

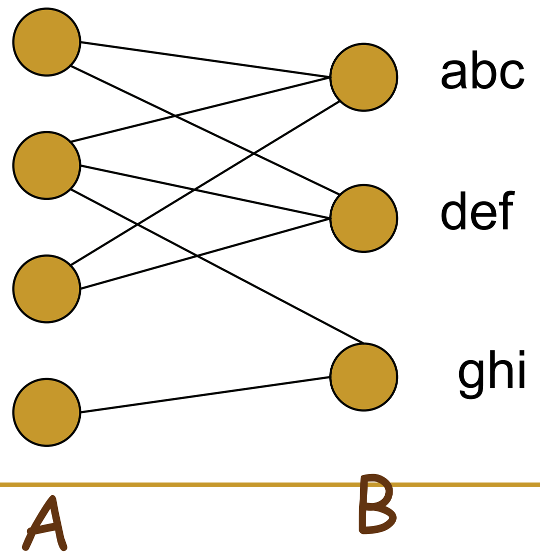
- $a_1 + 2\,a_2 + 3\,a_3 = b_1 + 2\,b_2 + 3\,b_3 = |S|$
  - Count the number of elements in each set
  - Each solution is a disjoint set cover



Comparison graph

- A bipartite graph (*A*, *B*, X), where
  - the vertices on either sides correspond to the sets in *A* and *B*, respectively
  - Edge between two sets that overlap (multiple edges if they overlap in many elements)





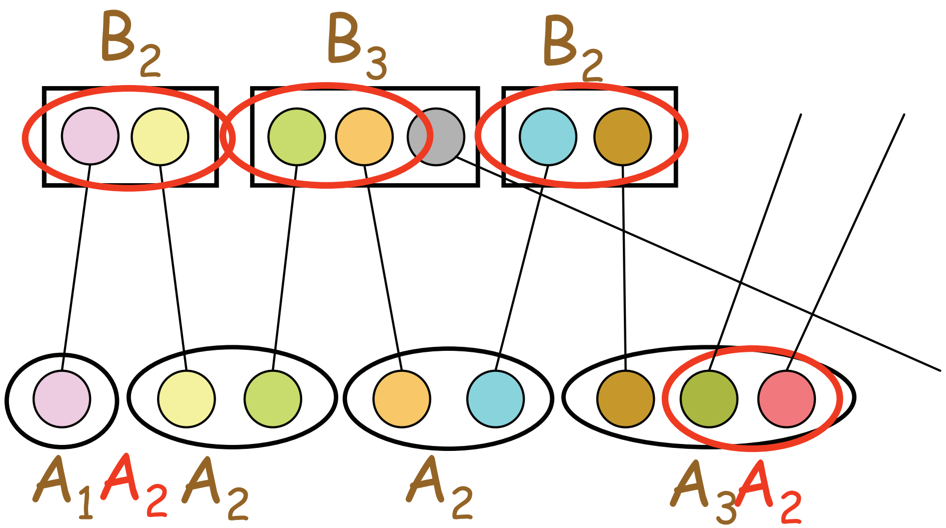
A component of comparison graph containing an A_1 –node:

- We shall show that it must have some restricted properties
- Cannot contain an A_3 node or another A_1
- Must have a matching A_1-node

Component containing an A_1 –node cannot contain an A_3 –node

- A path from A_1 -node to A_3 –node via B_2, B_3, A_2
  - Uses only 2 elements from each B_3 -node

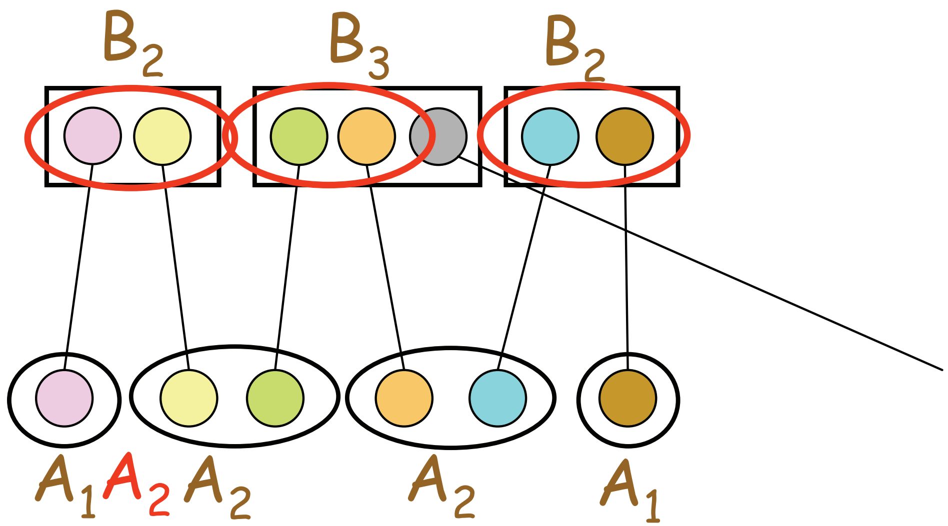
- Replace:
  A_1 & $A_2 \rightarrow B_2$
  $A_3 \rightarrow A_2$



- Reduces 3-sets → (0,1)-improvement

Component containing an A_1 –node cannot contain another A_1 –node

- A path from A_1 -node to another A_1-node.

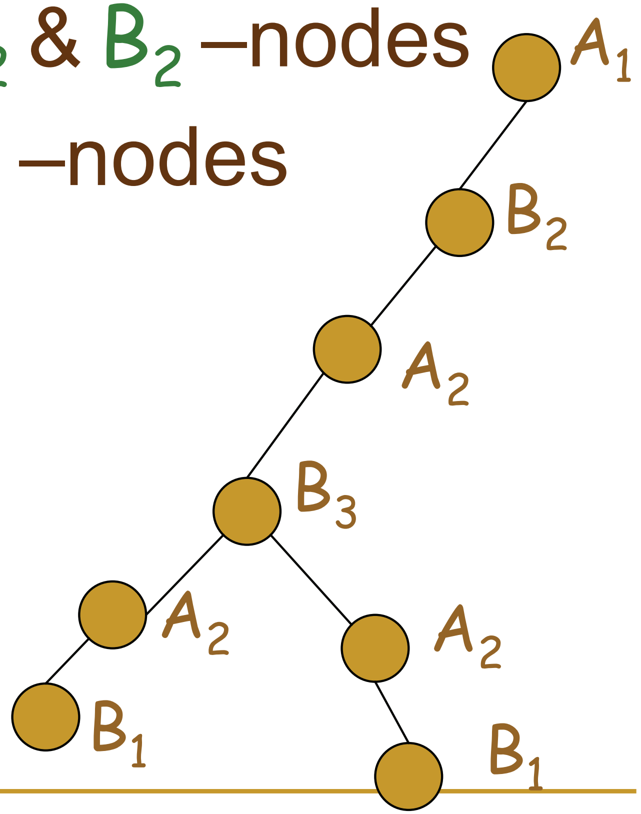


- Replace:
  A_1 & A_2 → B_2
  - Covers the same
  - Fewer sets
- ↳ (0,0)-improvement

Lemma 2:

- Component containing an A_1 –node is a tree.
  - Root: The A_1 –node
  - Internal nodes of degree 2: A_2 & B_2 –nodes
  - Internal nodes of degree 3: B_3 –nodes
  - Leaves: B_1 –nodes
- Therefore,

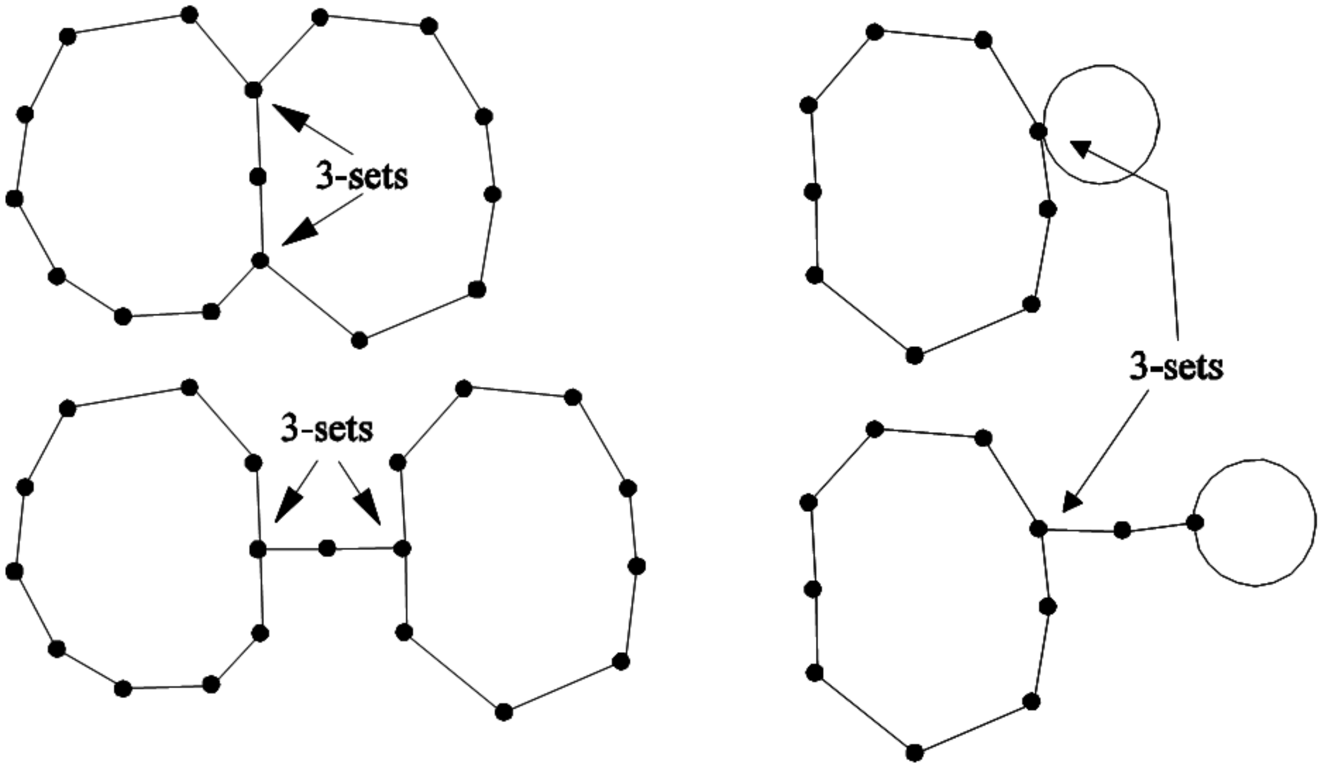
$$a_1 \leq b_1$$



Auxiliary graph

- Graph G= (B, A-A_3)
  - Vertex for each set in B
  - Edge for each set in A - A_3 = A_1 + A_2
- Thus, there is an edge between two sets in B, if there is an A_2-set that contains elements from both of them

Binocular

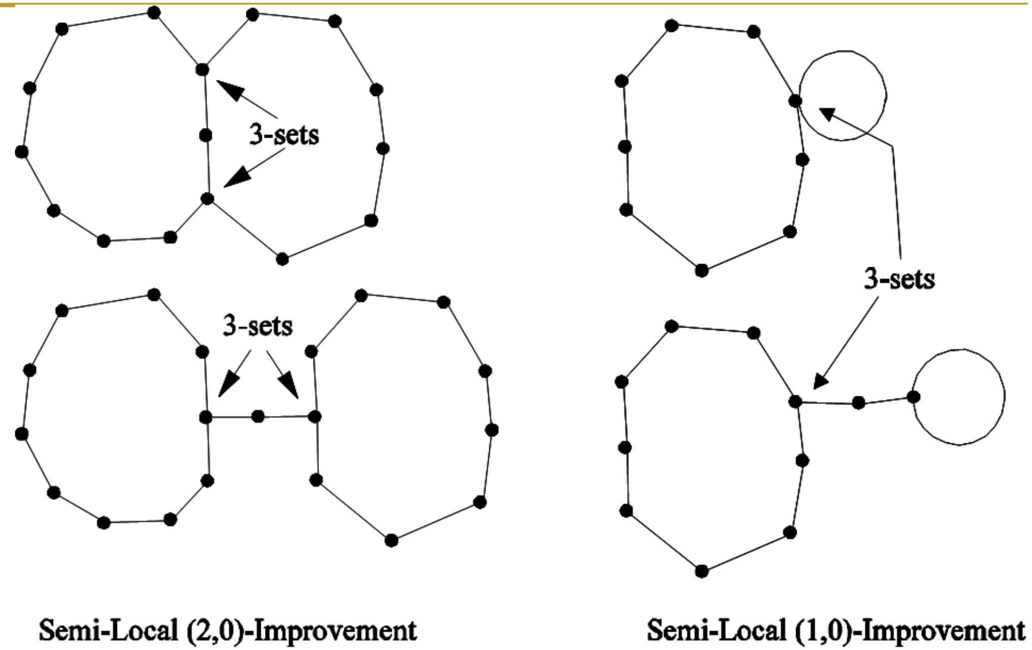


Semi-Local (2,0)-Improvement

Semi-Local (1,0)-Improvement

- A *binocular* is a subgraph that contains more than one cycle

Lemma 3



- Auxiliary graph with a binocular
  → (2,1)-improvement
- $\Rightarrow$ # edges in each component $\leq$ #vertices
- Namely,
  - $a_1 + a_2 \leq b_1 + b_2 + b_3$

Proof summary

- We derived 3 inequalities:
  - Obs 1: $a_1 + 2\,a_2 + 3\,a_3 = b_1 + 2\,b_2 + 3\,b_3$
  - Lemma 2: $a_1 \leq b_1$
  - Lemma 3: $a_1 + a_2 \leq b_1 + b_2 + b_3$
- Adding the inequalities,

$$3\,a_1 + 3\,a_2 + 3\,a_3 \leq 3\,b_1 + 3\,b_2 + 4\,b_3$$

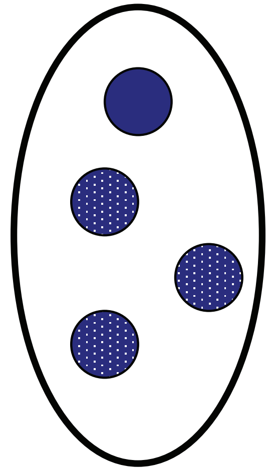
we get a strengthening of the theorem:

$$\boxed{3|A| \leq 4\,|B| - b_1 - b_2}$$

Back to Color Saving:
Assume G contains no 4-independent set

- Here:
  - $|S| = n = b_1 + 2\,b_2 + 3\,b_3$
  - $B = \chi$, A = #colors (used by algorithm)
- We have:
  - $5n\text{-}5b = 2(3n\text{-}4b+b_1+b_2) + (3b\text{-}2b_1+b_2\,\text{-}n)+ b_2$
    $\geq 2(3n - 3a) + 0 + 0$
- So,
  - $(n\text{-chi})/(n\text{-\#colors}) = (n\text{-}b)/(n\text{-}a) \leq 6/5$

Color Saving



- For graphs with 4-IS and larger
  - We greedily color 4-sets as possible.
  - For each such set
    - Algorithms saves 3 colors
    - Optimal solution saves at most 4 colors
    - Ratio of 4/3.
- Refined analysis of Duh/Furer:
  - Ratio 360/289 ≈ 1.246

Summary

- Semi-local search: Matching + LS
  - 4/3-ratio for 3-Set Cover
  - $H_k - \frac{1}{2}$ for k-Set Cover, using greedy rounds
  - 360/289-ratio for Color Savings
- Open questions
  - Improve the ratio $H_k - \frac{1}{2}$
    - Combine Greedy rule with local search

Part III: Independent Set in Hypergraphs

How good is greediness for another SetCover equivalent

[H, Elena Losievskaja, 2006]

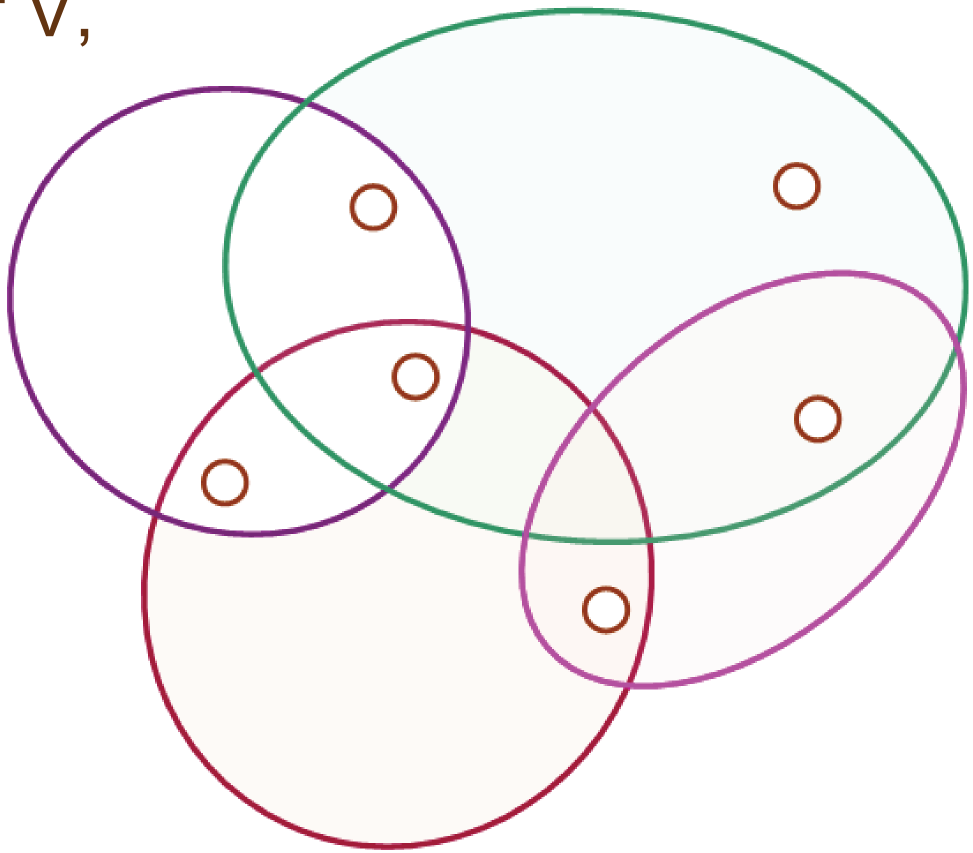
Definitions

A hypergraph H is a pair (V,E):

V is a discrete set of vertices,
E is a collection of subsets of V,
or (hyper)edges.

Graphs are hypergraphs with all edges of size 2.

Degree of a vertex v is the number of incident edges:
$d(v) = |\{ e \in E : v \in e \}|$

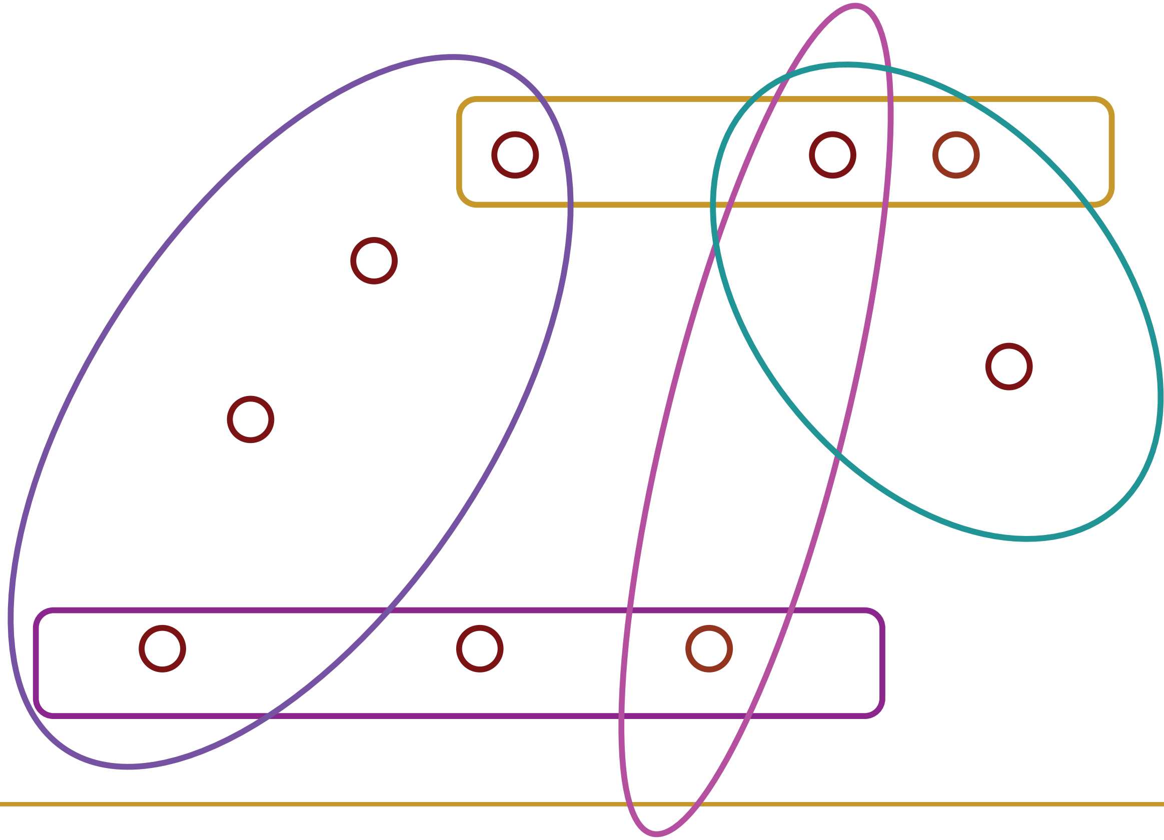


Independent Set

A (weak) independent set in a hypergraph is a subset of vertices that contains no edge.

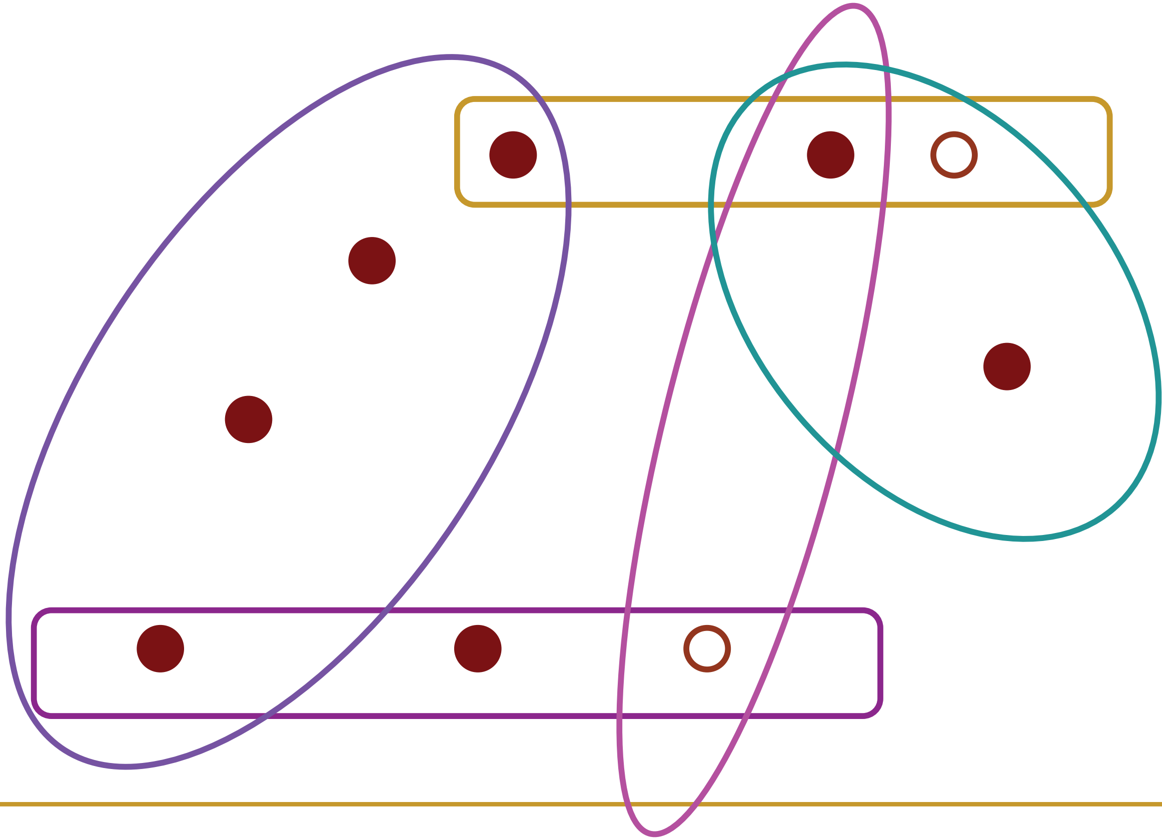
Independent Set

A (weak) independent set in a hypergraph is a subset of vertices that contains no edge.



Independent Set

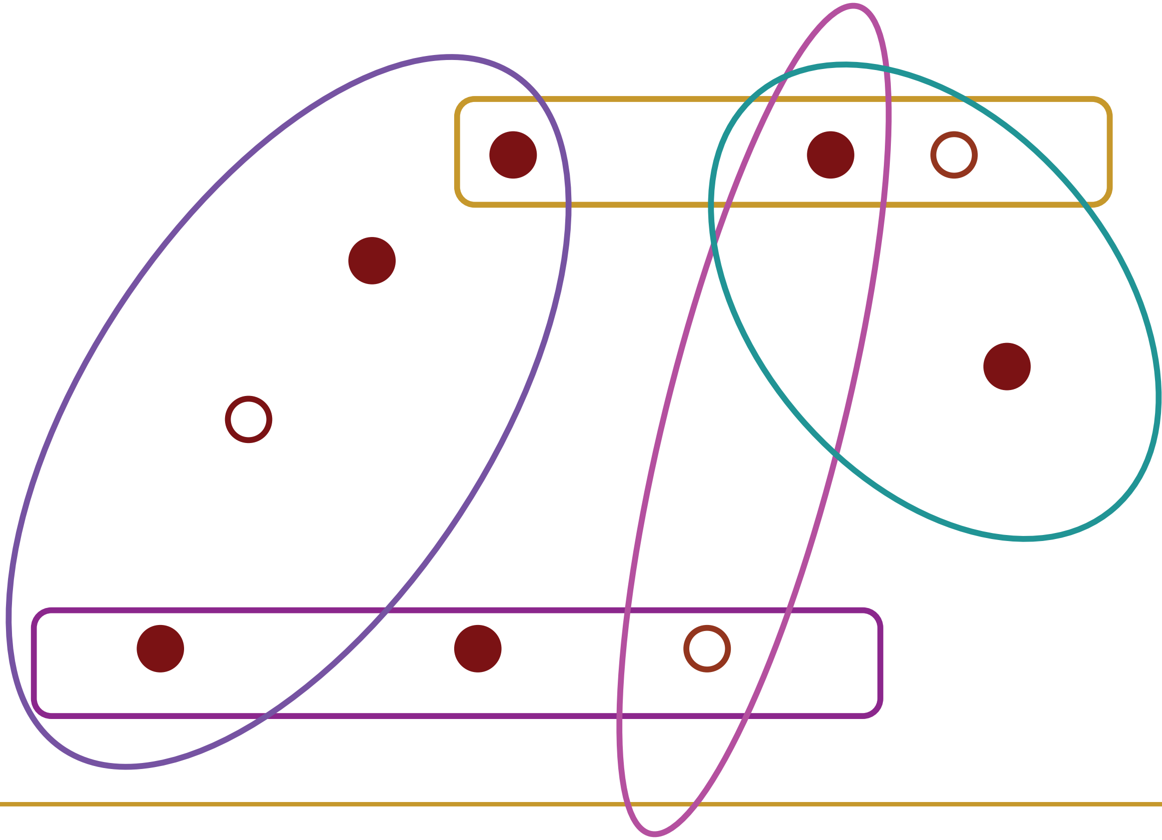
A (weak) independent set in a hypergraph is a subset of vertices that contains no edge.



Not an independent set

Independent Set

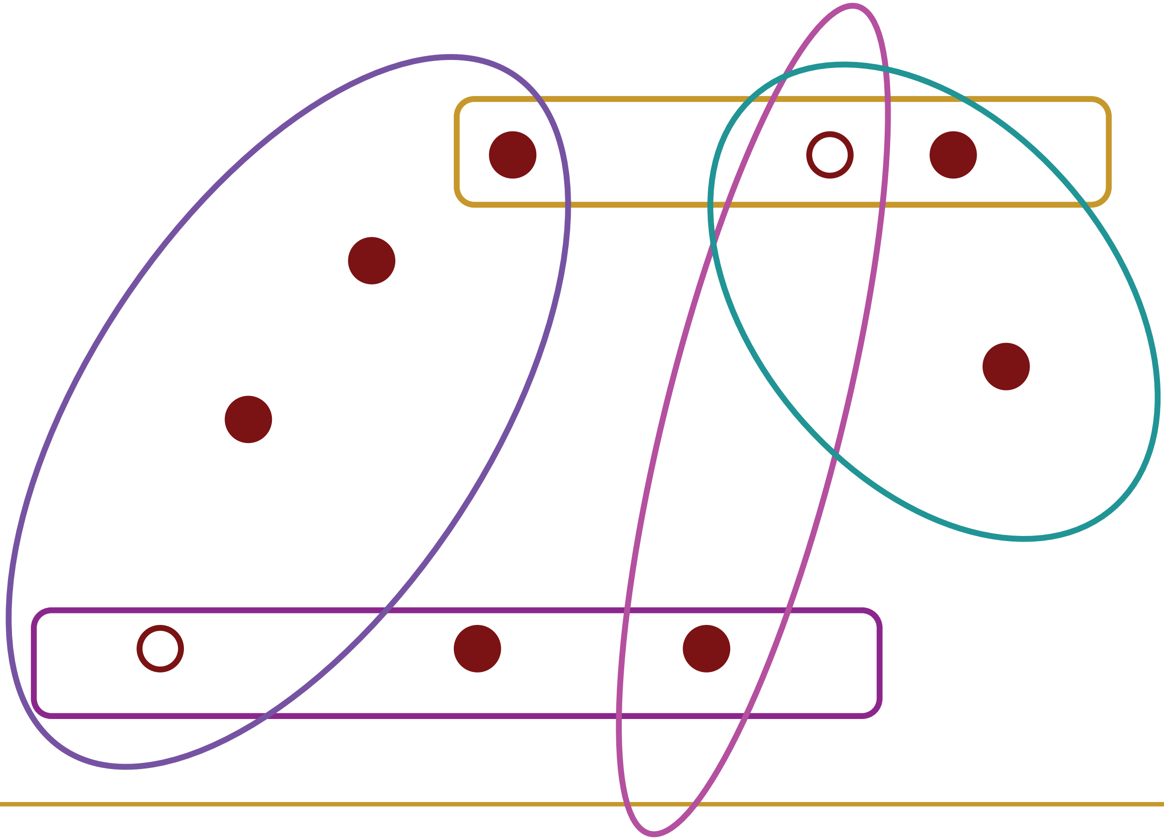
A (weak) independent set in a hypergraph is a subset of vertices that contains no edge.



Size = 6

Independent Set

A (weak) independent set in a hypergraph is a subset of vertices that contains no edge.



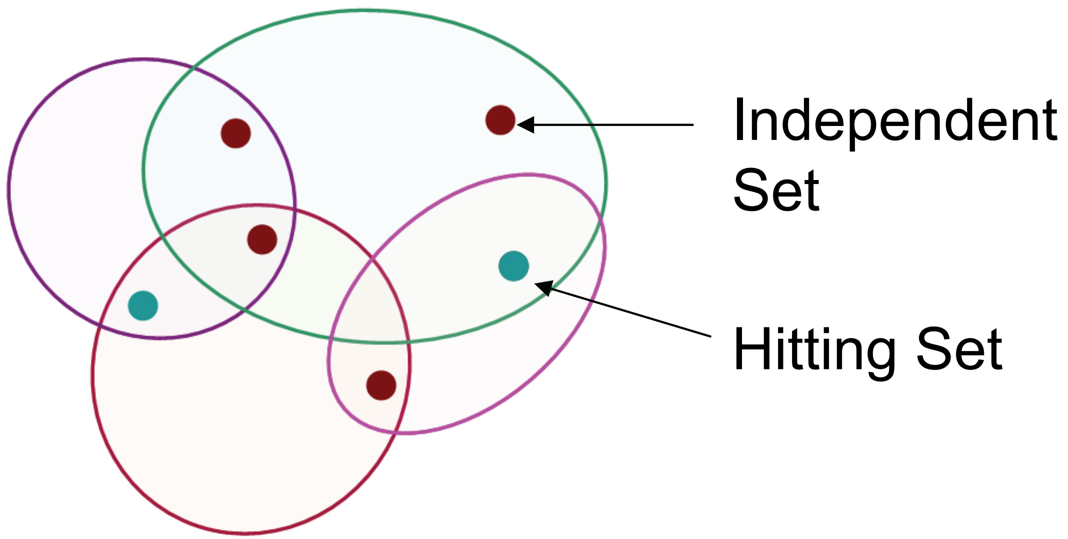
Size = 7

Independent Set

The problem of finding maximum independent set is strongly related to several other important problems:

Hitting Set ⇝ Independent Set

Hitting Set problem:
given a hypergraph,
find the smallest subset of vertices that covers every edge

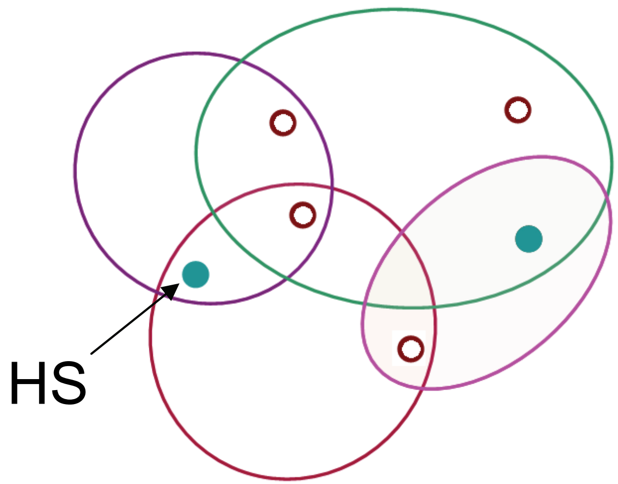


Hitting Set ≡ Set Cover

Set Cover problem:

given a universe of elements and a collection of sets,

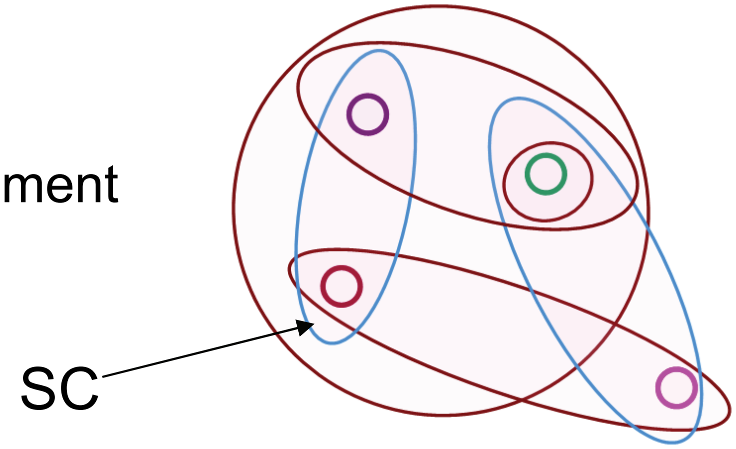
find the smallest subcollection of sets that covers every element in the universe



HS ↔ SC

edge ↔ element

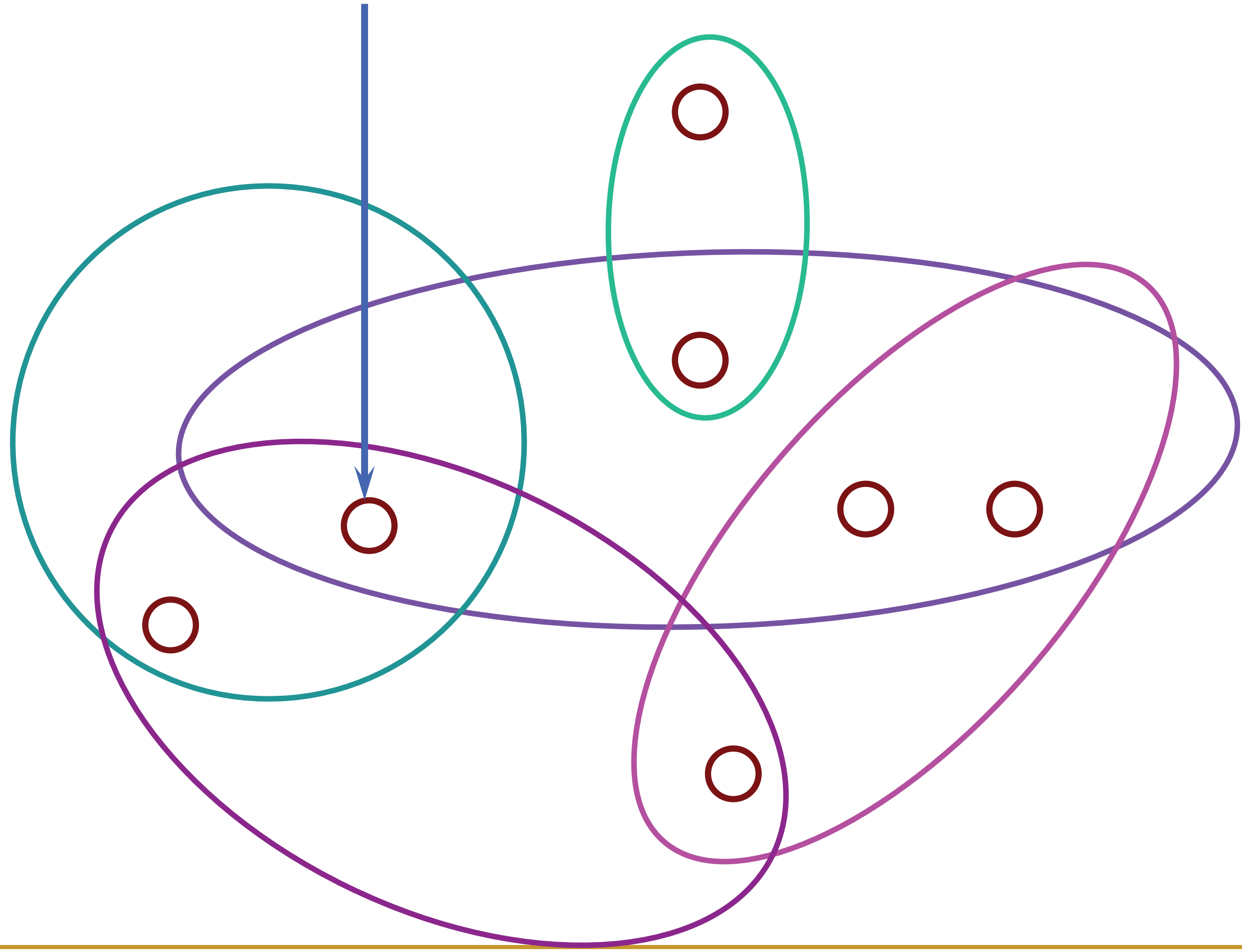
vertex ↔ set



In terms of exact optimization all three problems, Independent Set, Hitting Set and Set Cover, are equivalent.

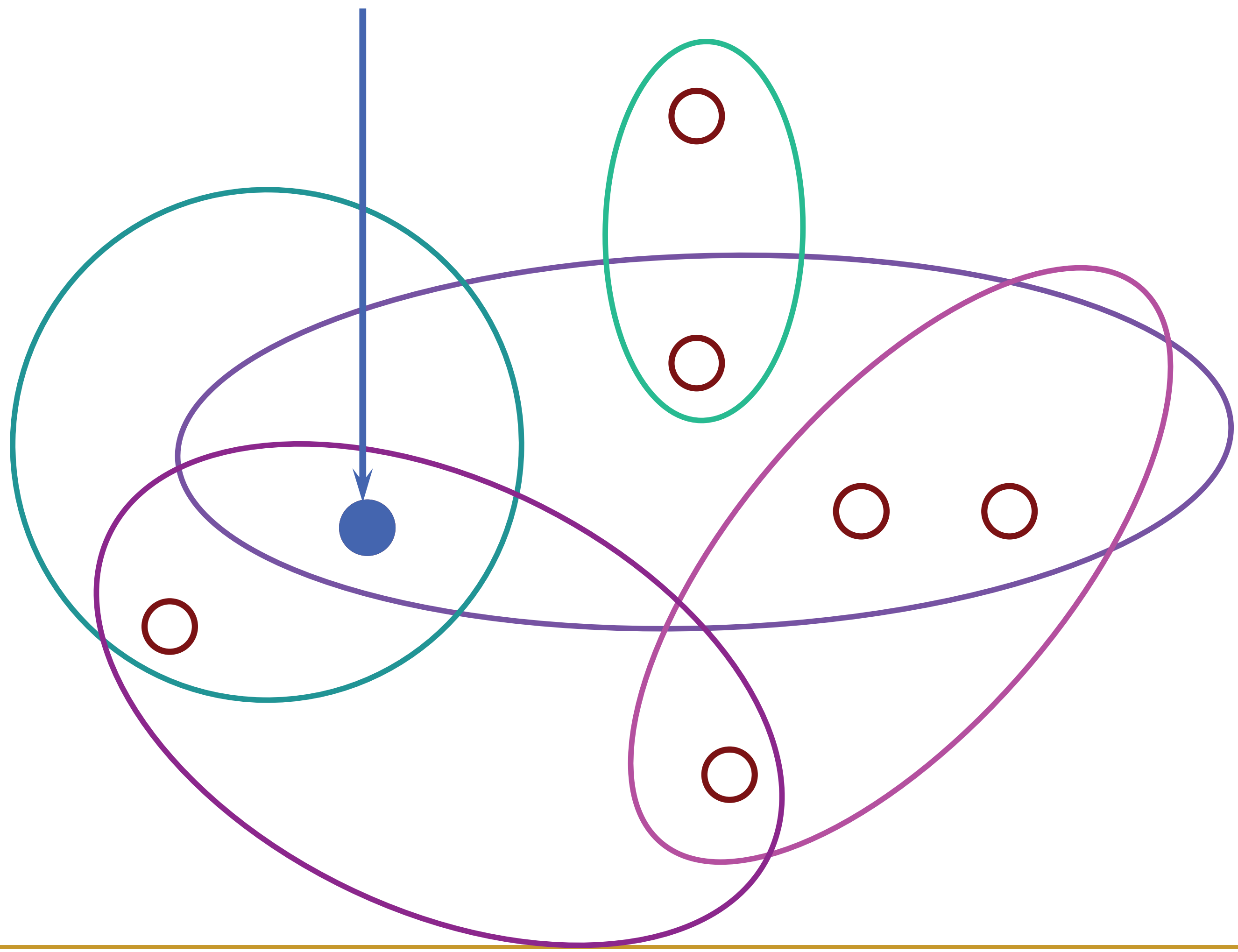
GreedyMAX

1. Select a vertex of **maximum** degree



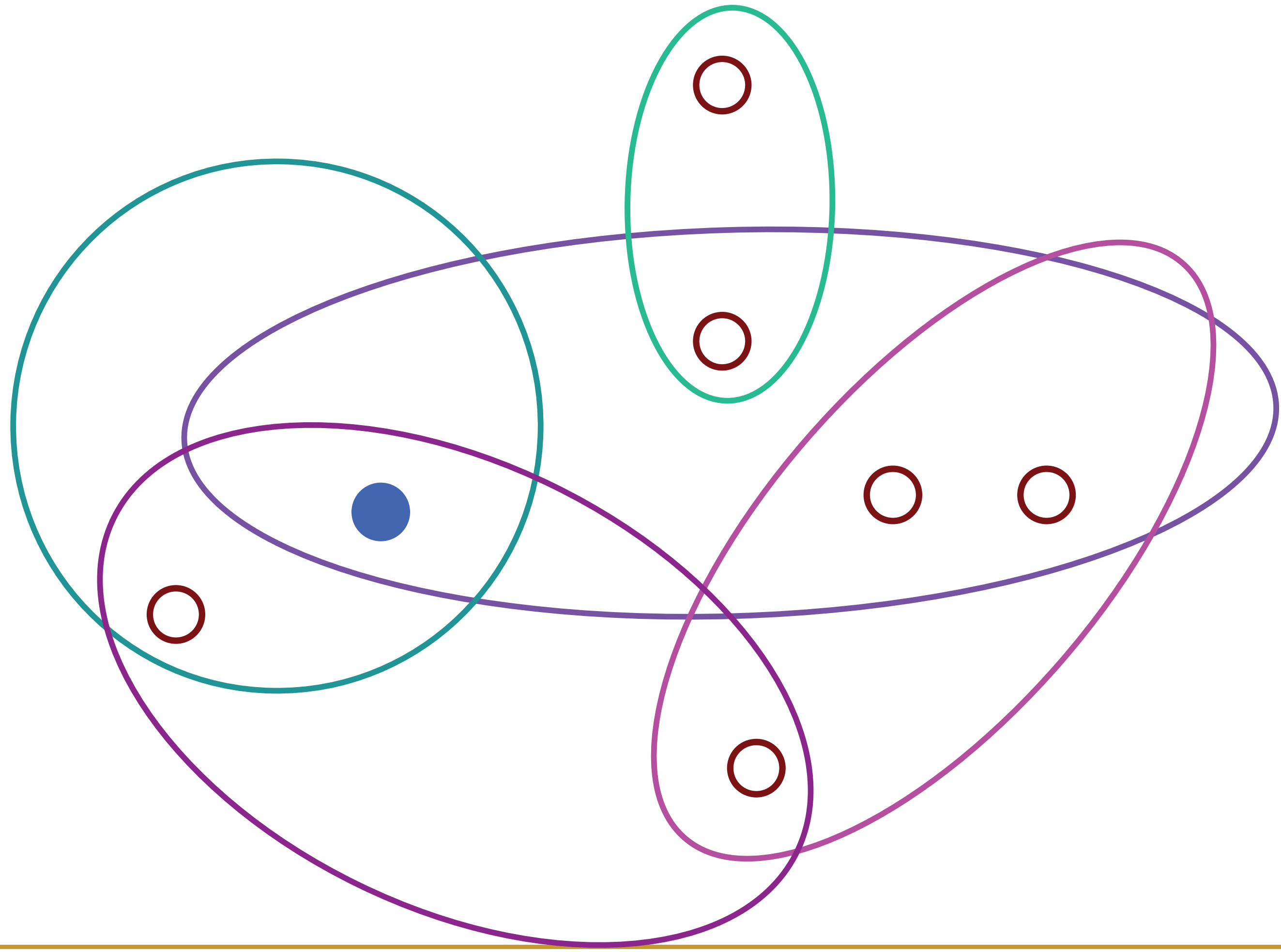
GreedyMAX

2. Add the vertex to the cover S



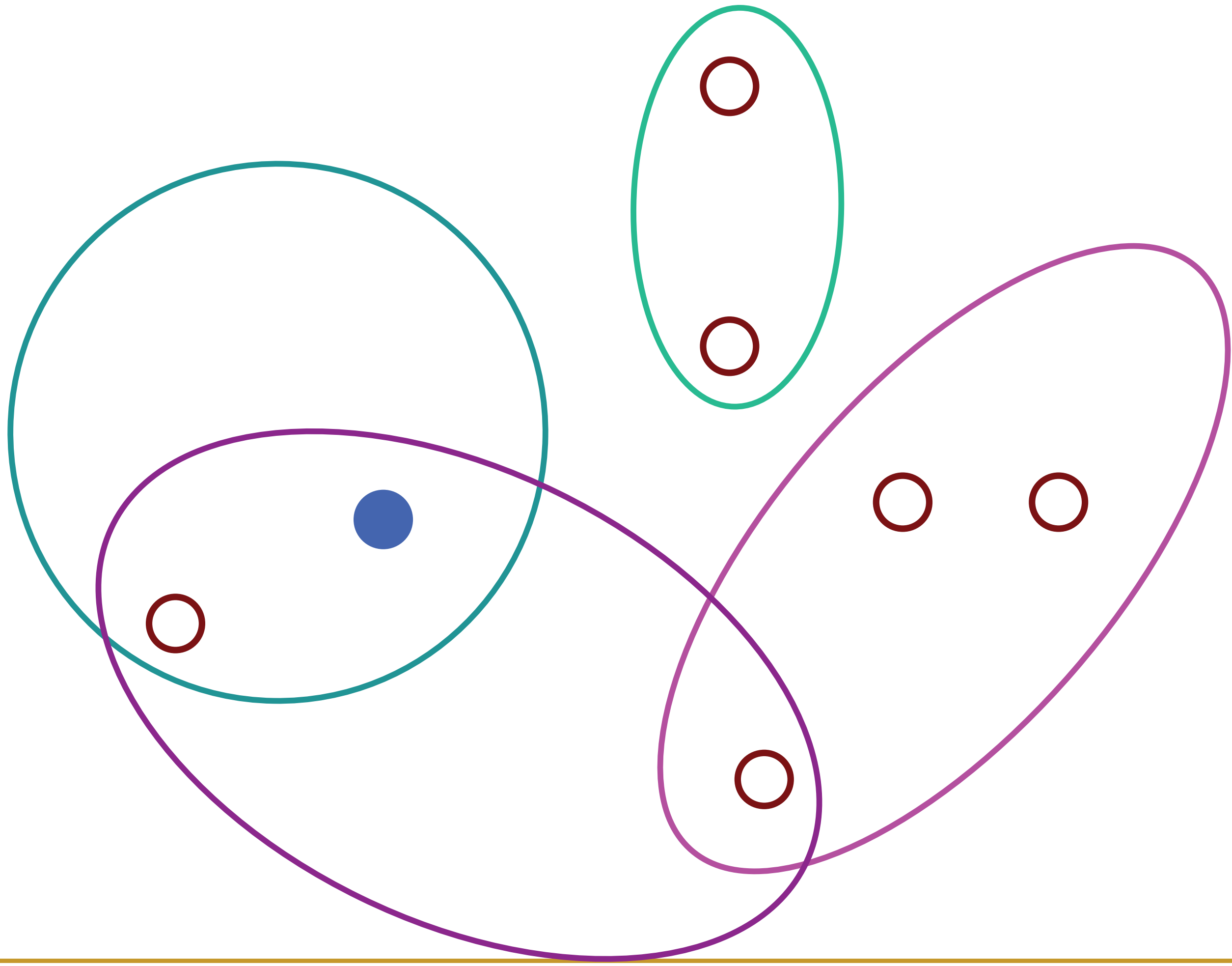
GreedyMAX

3. Delete the vertex along with all incident edges



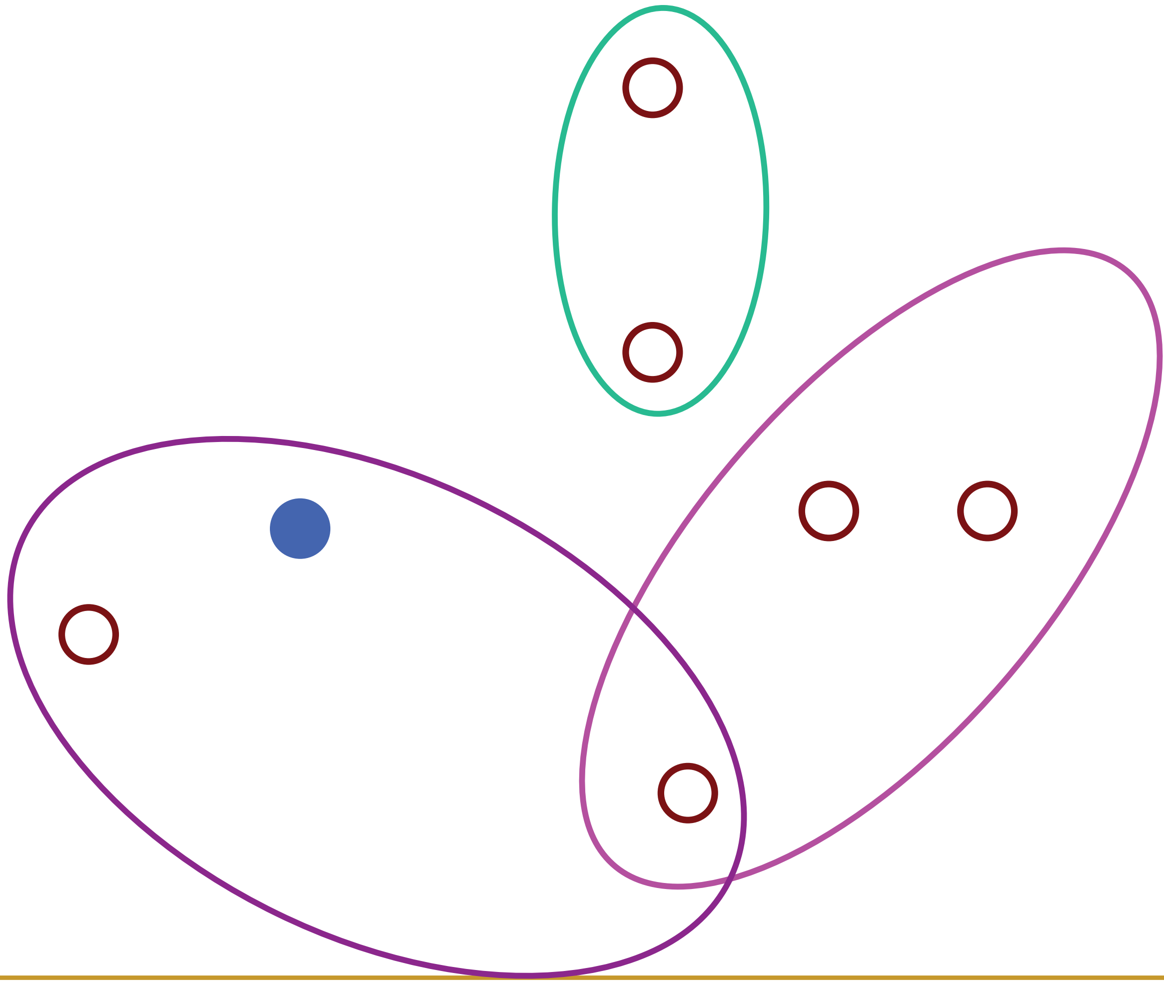
GreedyMAX

3. Delete the vertex along with all incident edges



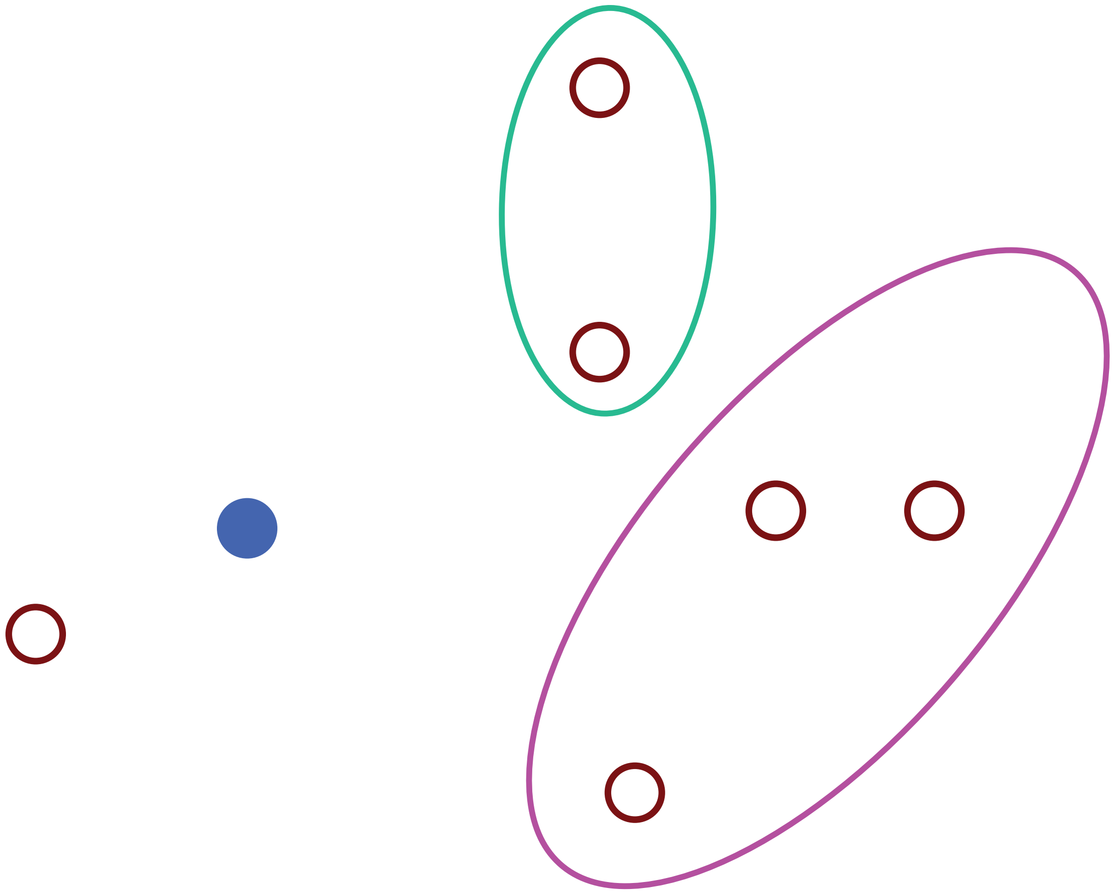
GreedyMAX

3. Delete the vertex along with all incident edges



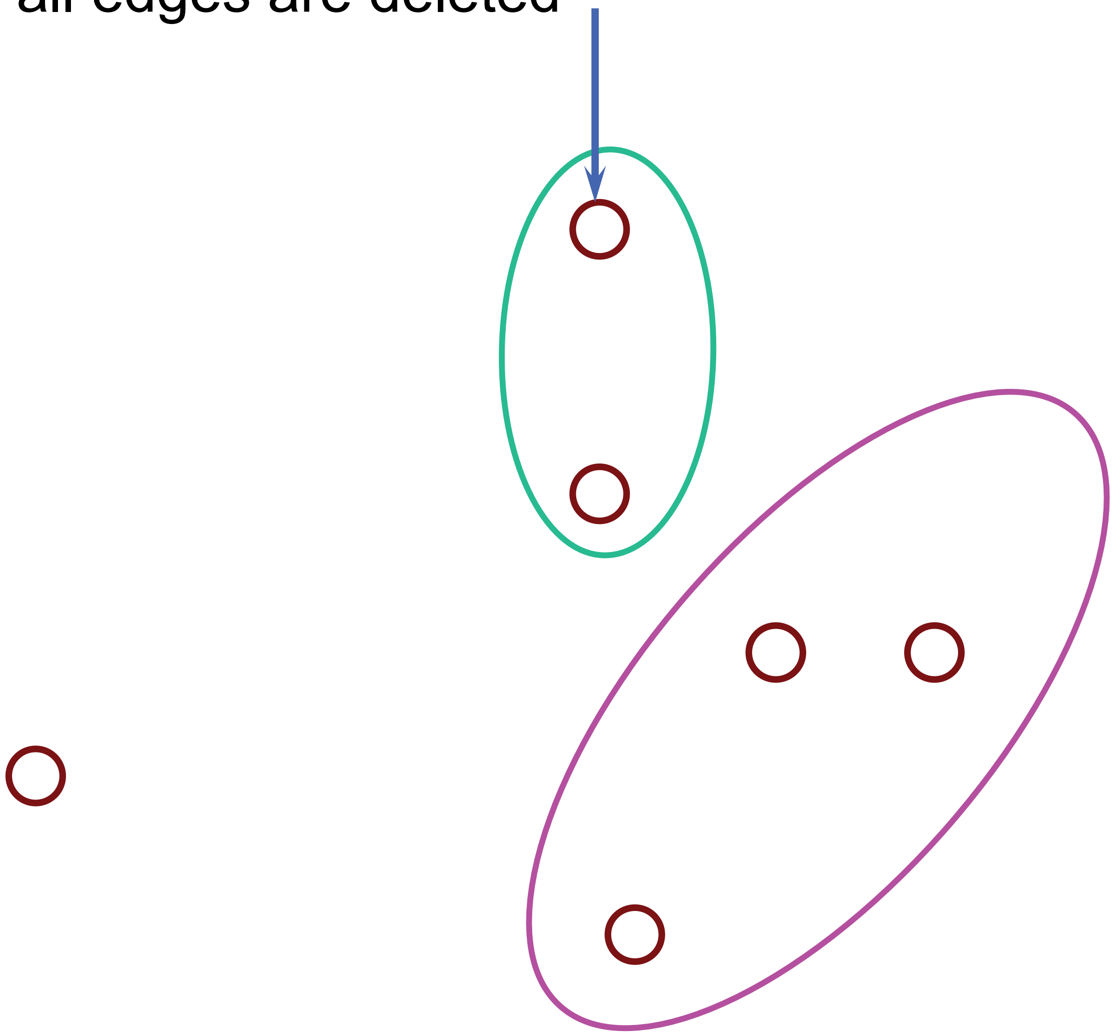
GreedyMAX

3. Delete the vertex along with all incident edges



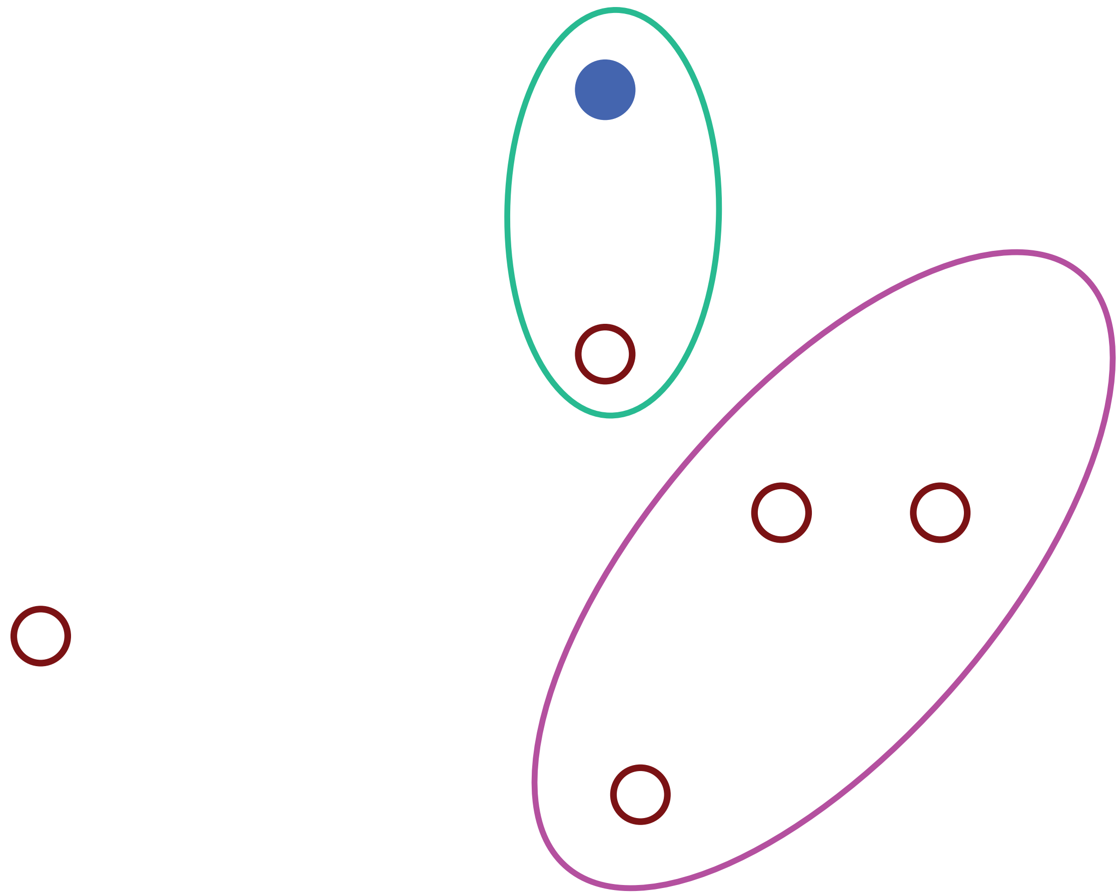
GreedyMAX

Iterate until all edges are deleted



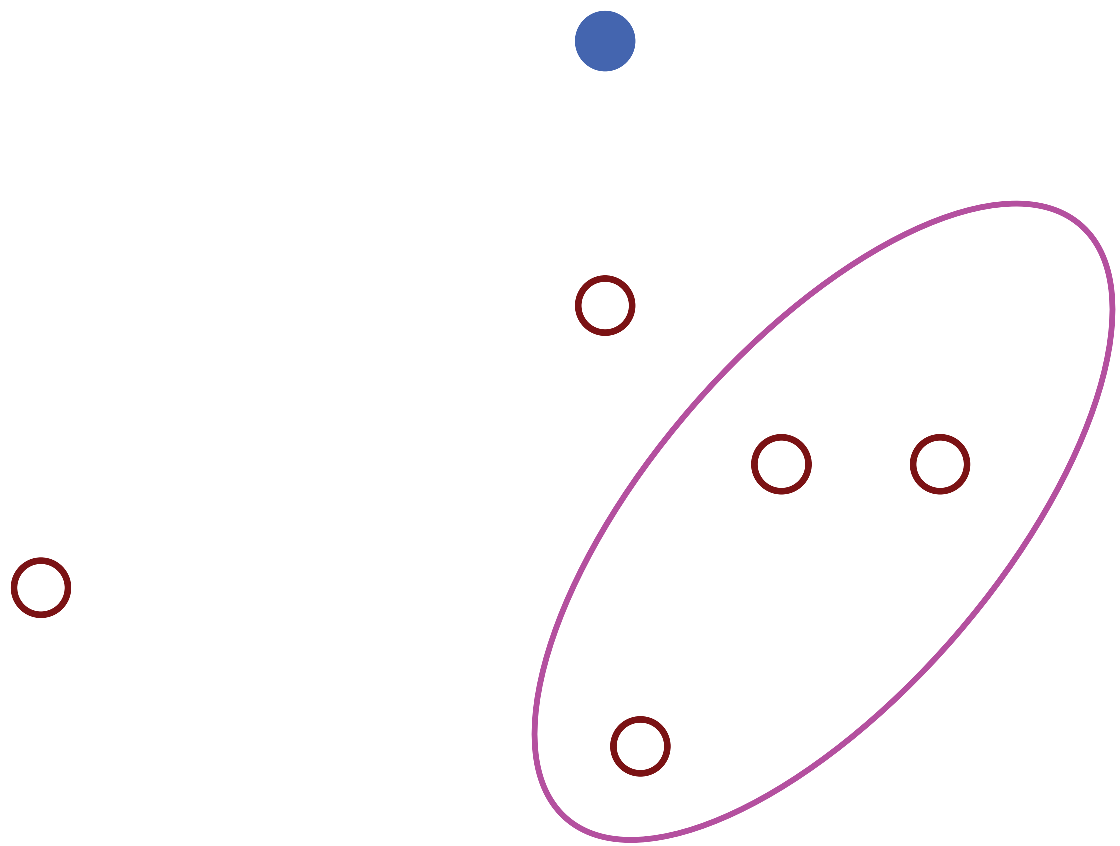
GreedyMAX

Iterate until all edges are deleted



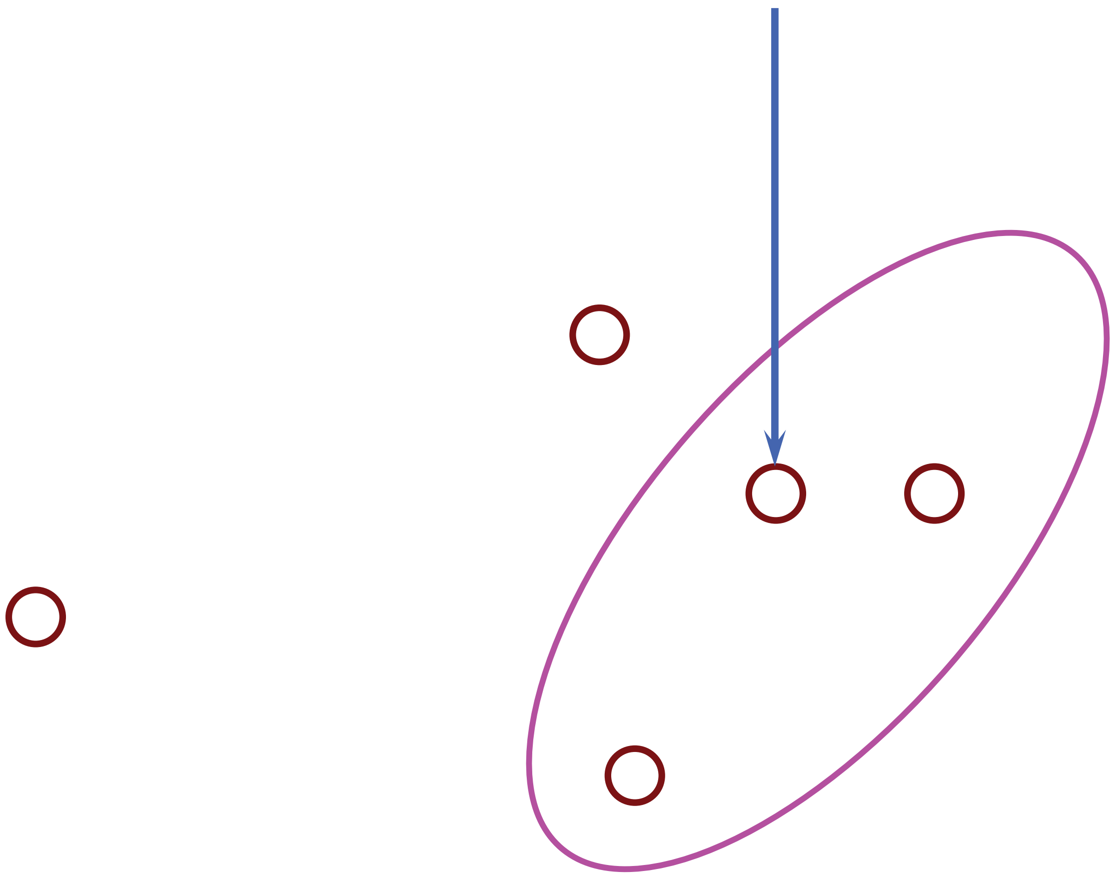
GreedyMAX

Iterate until all edges are deleted



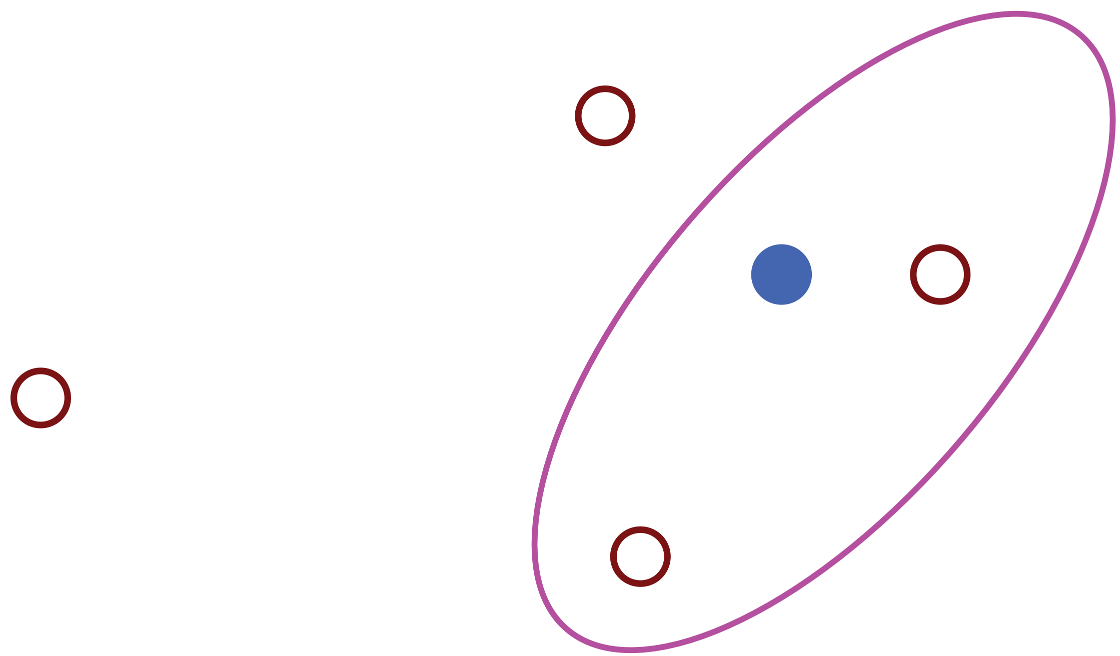
GreedyMAX

Iterate until all edges are deleted



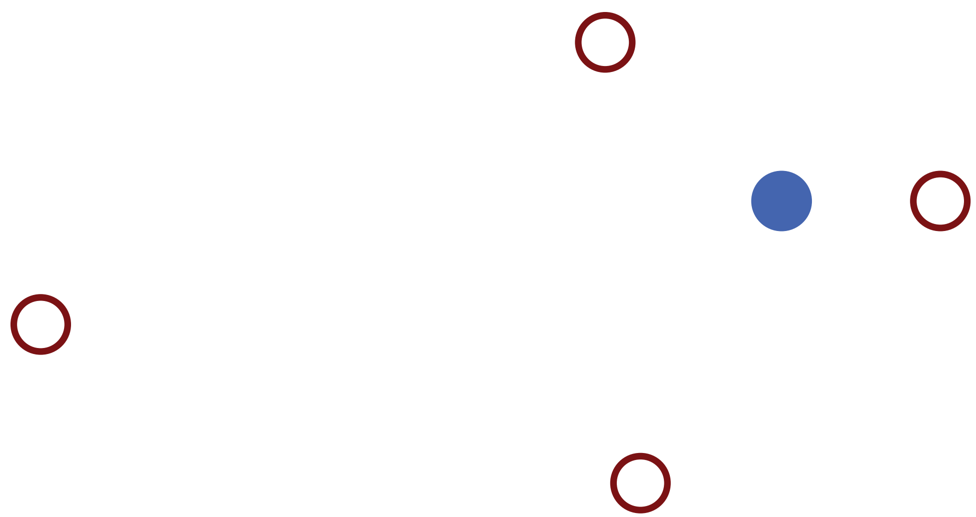
GreedyMAX

Iterate until all edges are deleted



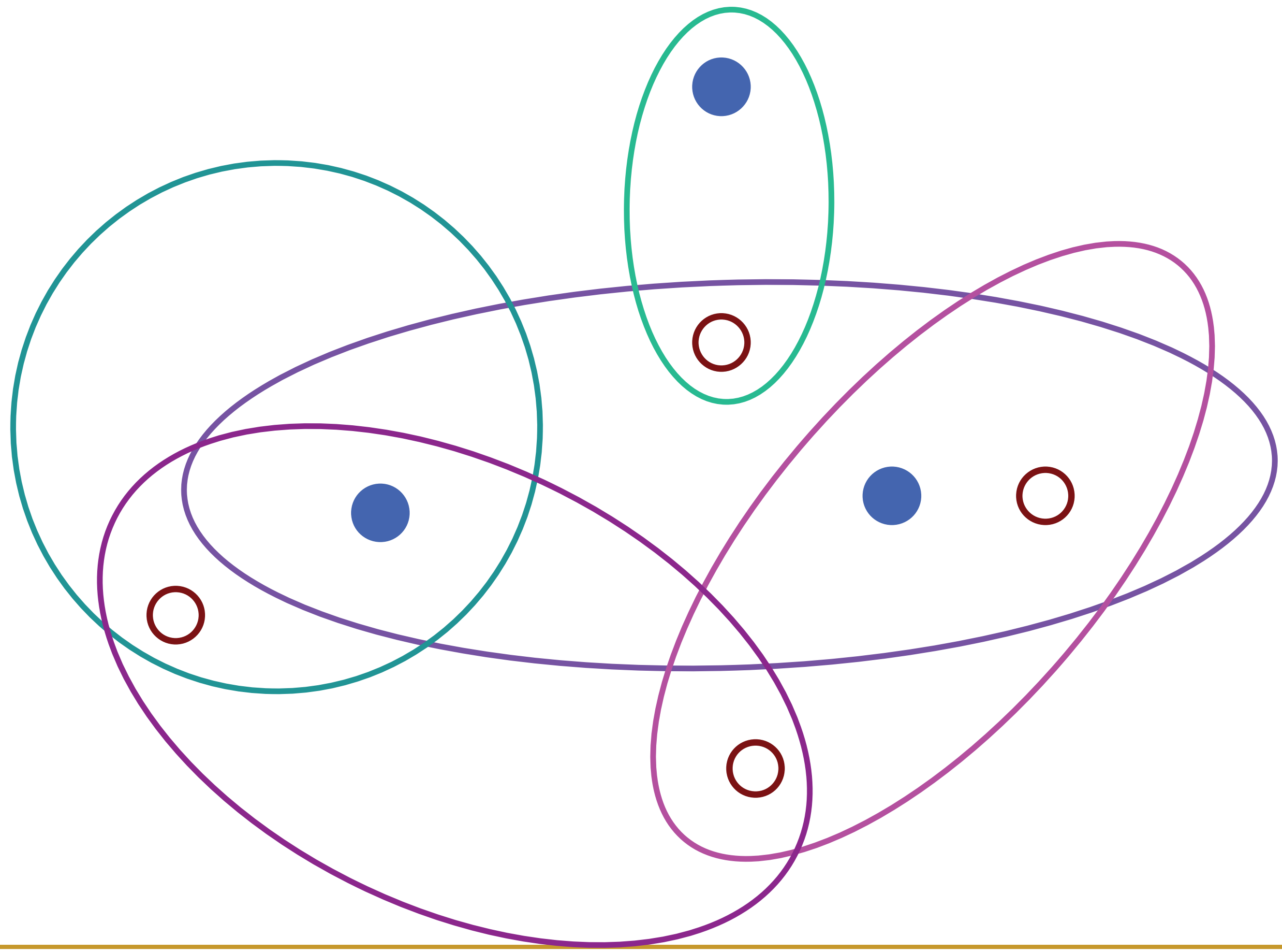
GreedyMAX

Iterate until all edges are deleted



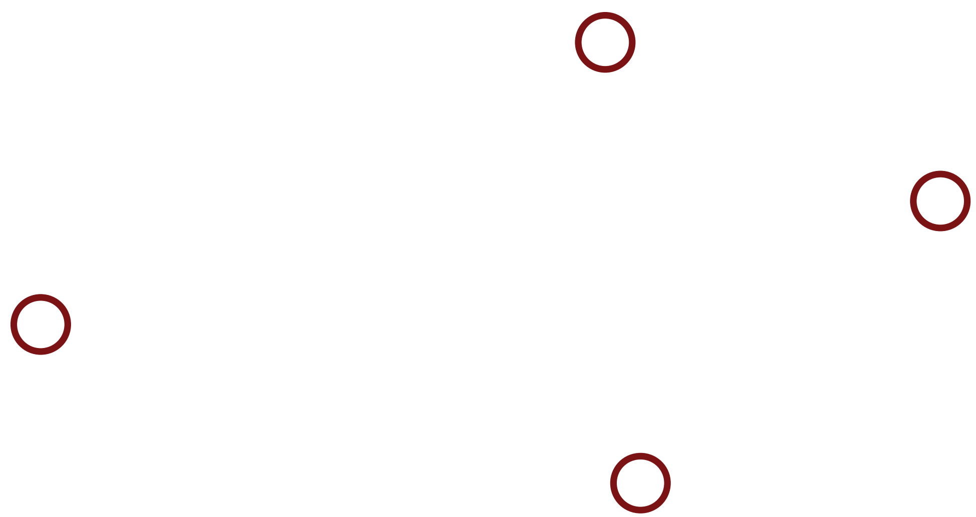
GreedyMAX

The vertices in S form a hitting set (cover)



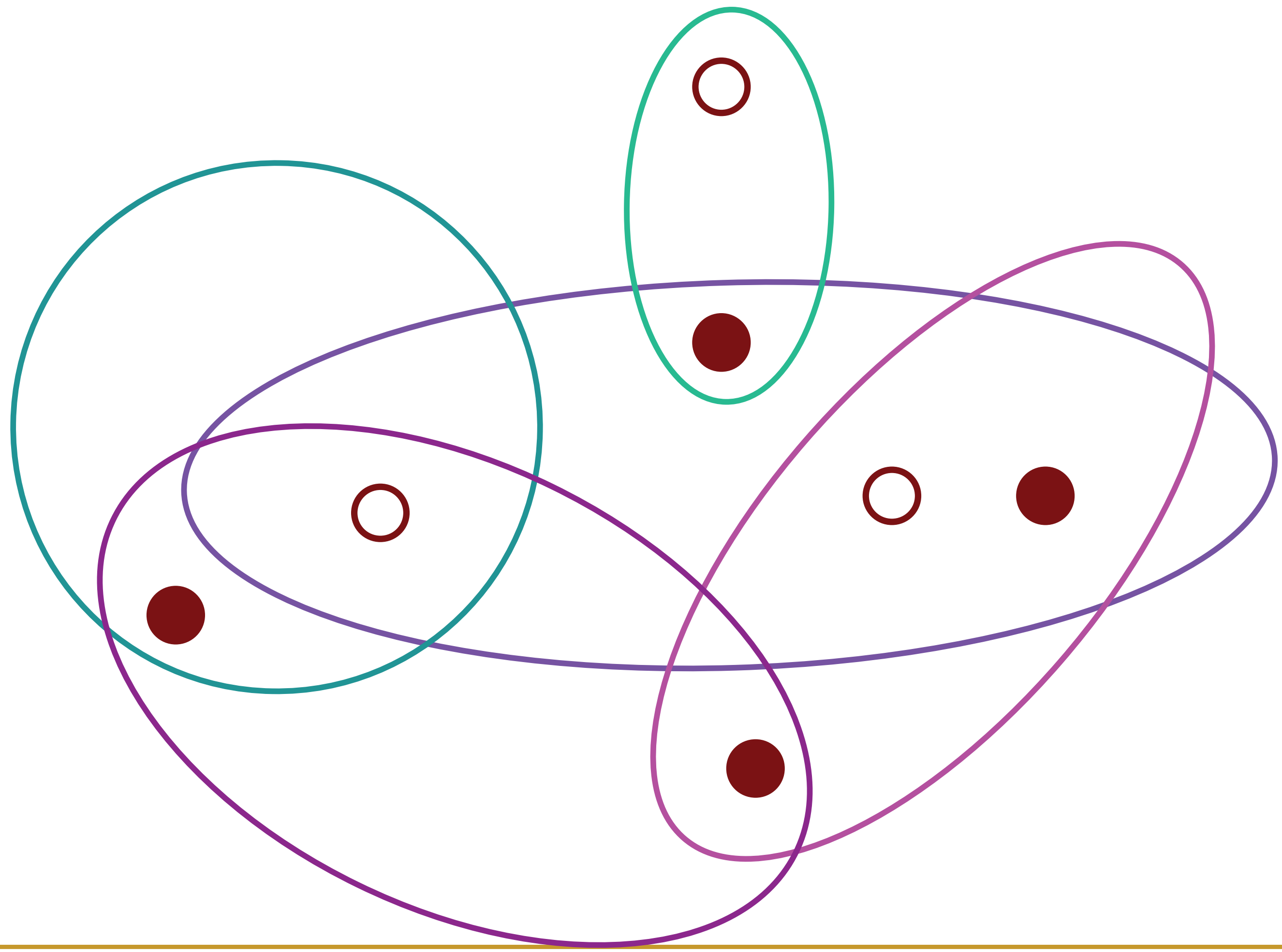
GreedyMAX

The independent set I found is V-S,
the vertices not in the cover S



GreedyMAX

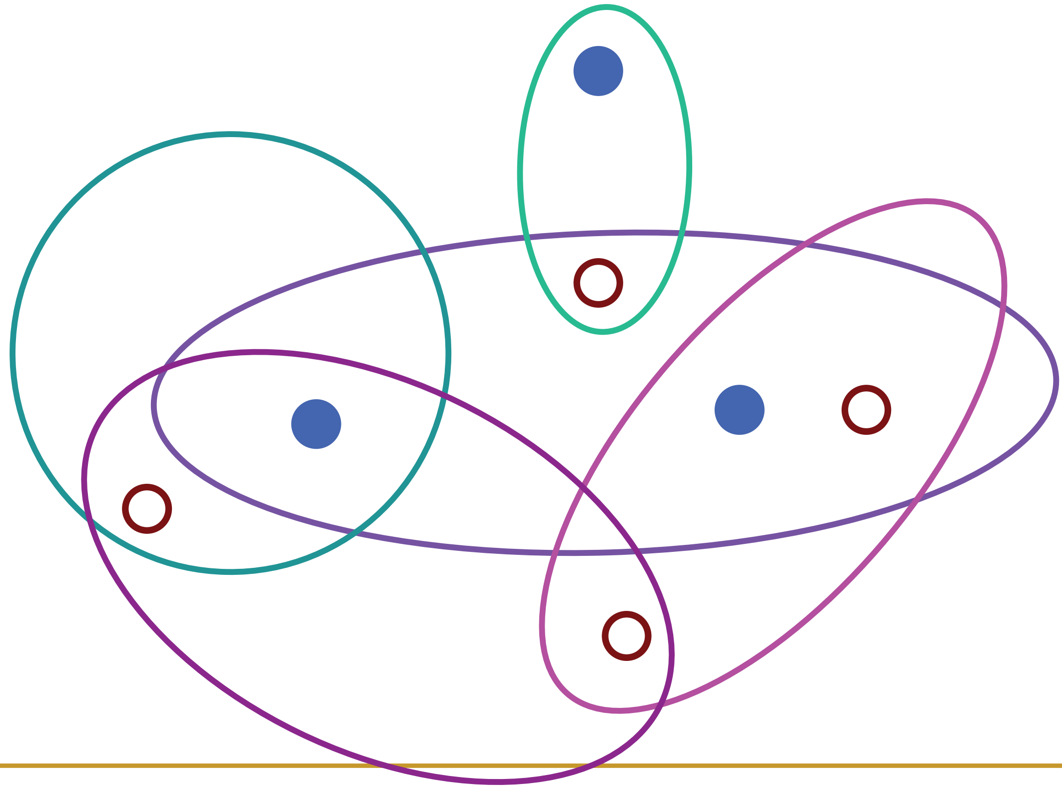
The independent set I found is V-S, vertices NOT in the cover S



GreedyMAX = GreedySetCover

(Vertex ⇔ set, edge ⇔ element) → GreedyMAX = GreedySetCover.

The GreedySetCover algorithm:
iteratively selects a set that covers the largest number of uncovered elements.



GreedyMAX for Set Cover problems

Results on the greedy set cover algorithm:

- Performance ratio $H_n \approx \ln n + 1$ (Johnson; Lovász)
- The best possible approximation algorithm for the Set Cover problem (Feige 1998), within lower order terms
- The best possible for various related problems:
    - Weighted Set Cover [Chvatal 1979]
    - Sum Set Cover [Feige,Lovasz,Tetali 2004]
    - Test Set []
    - Entropy Set Cover [Cardinal,Fiorini,Joret 2006]

GreedyMAX

Differential approximation ratio of GreedySetCover: $$\rho = \max_{\forall H} \frac{n - |S^*|}{n - |S|}$$

(i.e. we measure how many sets are **not** included in the cover)

Approximation ratio of GreedyMAX: $$\rho = \max_{\forall H} \frac{|I^*|}{|I|} = \max_{\forall H} \frac{n - |S^*|}{n - |S|}$$

where I*, I – an optimal and greedy independent sets

S*, S – an optimal and greedy covers

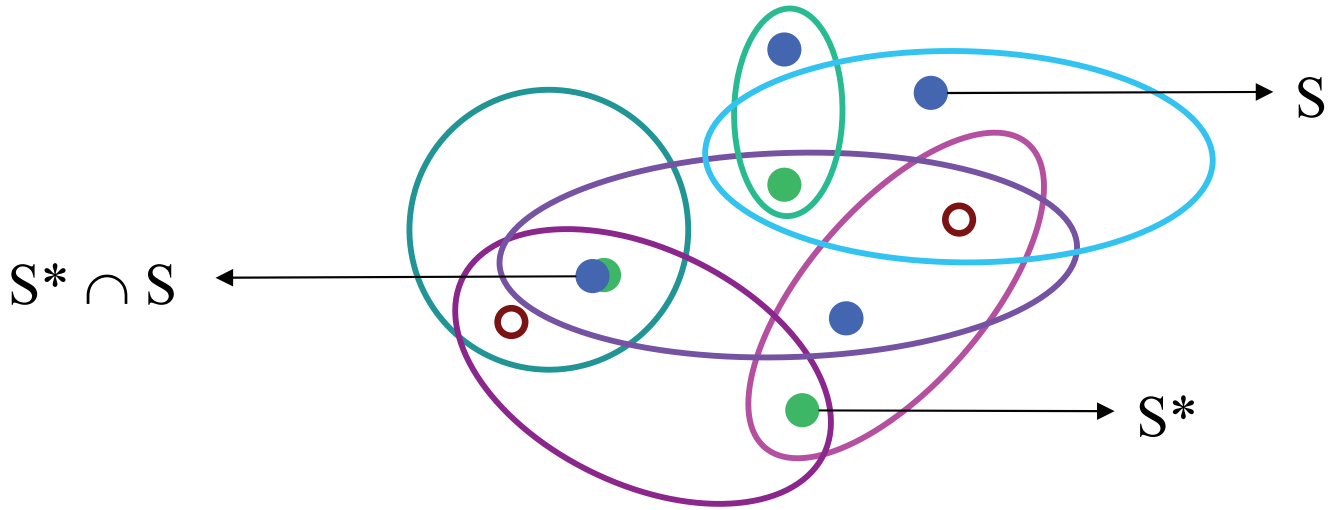
Bazgan, Monnot, Paschos and Serrière[1]:

- GreedySetCover: $\frac{\Delta}{1.365} \le \rho \le \frac{\Delta + 1}{4}$
- Local search: $\rho = \frac{\Delta + 1}{2}$

GreedyMAX

Bazgan, Monnot, Paschos and Serrière 2005:

- <u>Main algorithm</u>: GreedyMAX $\rightarrow$ a greedy cover;
- <u>Post processing</u>: exclude redundant vertices from the cover $\rightarrow$ a **maximal** greedy cover;
- <u>Analysis</u>:
  - compare how optimal and greedy vertices cover the edges in the hypergraph



Complicated analysis (sketch)

We started by extending the analysis of Bazgan et al:

- Count incidences of all vertices from $S \setminus S^*$, $S^* \setminus S$, $S^* \cap S$, $\overline{S^* \cup S}$ in all edges, obtaining edgesets E_1, E_2, E_3
- Use variables k, $l \in [0,1]$ and average degree of vertices to express the dependence between $S \setminus S^*$, $S^* \setminus S$, $S^* \cap S$, $\overline{S^* \cup S}$
- Bound the approximation ratio by a single multivariable function:

$$\rho \leq \frac{b\bar{d}_b - E_1 - (1-l)E_3 - \frac{1-k}{\Delta-1}\left(\Delta E_3 + lE_2\right) + \frac{k}{\bar{d}_r(\Delta-1)}\left(\Delta E_3 + lE_2\right)}{b + \frac{k}{\bar{d}_r(\Delta-1)}\left(\Delta E_3 + lE_2\right)} = f\left(k, l, \bar{d}_b, \bar{d}_r\right)$$

Complicated analysis (sketch)

- Find the maximum of $f\left(k,l,\bar{d}_b,\bar{d}_r\right)$ by
  - using variables *x*, *y*, *s* $\in$ [0,1] to bound the dependence between E_1, E_2, E_3
  - expressing $\bar{d}_b = g(\Delta)$, $\bar{d}_r = h(\Delta)$
  - taking partial derivatives $\frac{\partial f}{\partial k}$, $\frac{\partial f}{\partial l}$, $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial f}{\partial s}$

Eventually, we obtain a tight ratio of $\rho \le \frac{\Delta+1}{2}$

Weaknesses:
- Proof too long and complicated
- Requires post-processing phase to ensure maximality of IS

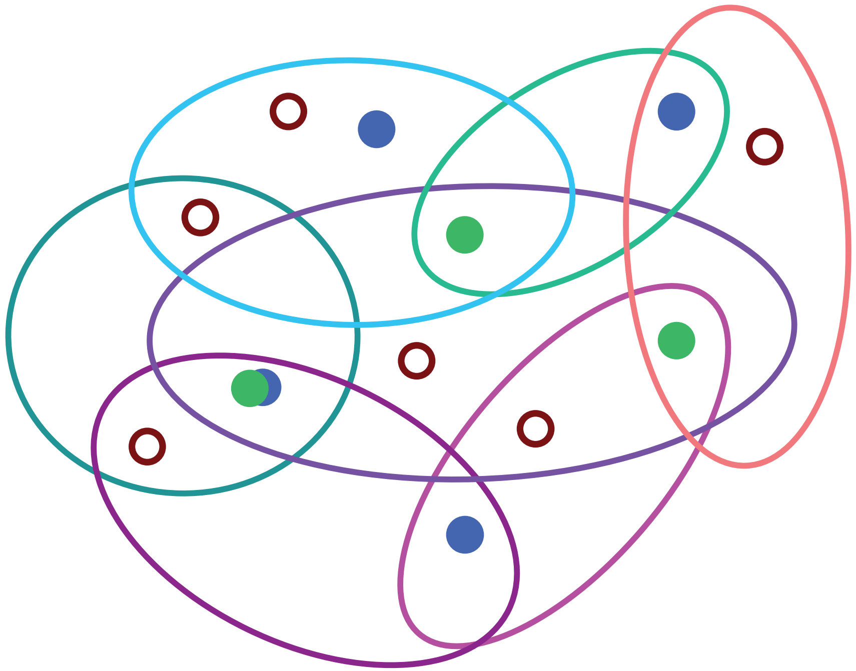
Simpler proof

A much simpler proof:

1. The “hardest” hypergraphs for GreedyMAX are ordinary graphs.
2. GreedyMAX in graphs has ratio $\rho \le \frac{\Delta+1}{2}$.

GreedyMAX

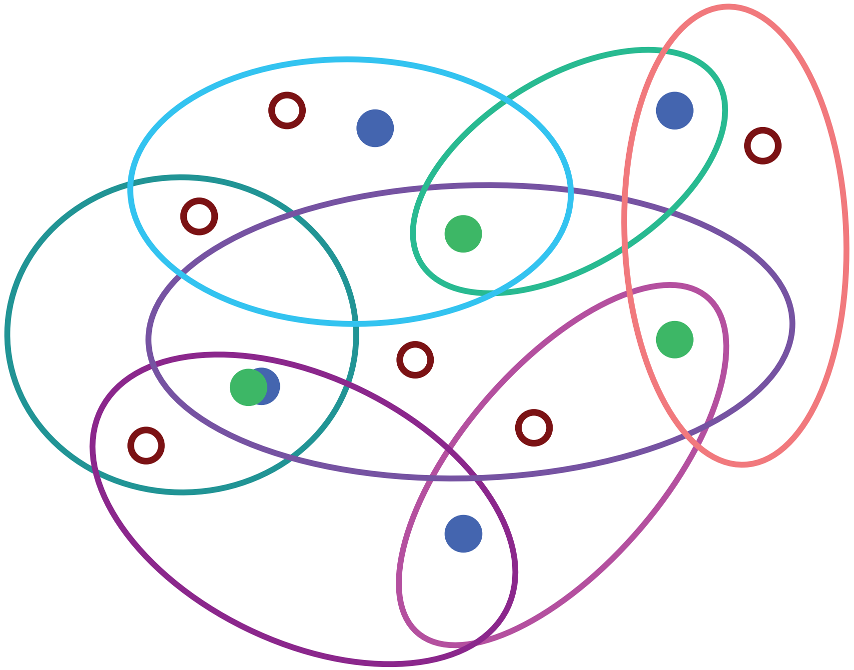
We "shrink" the hypergraph H to a graph G



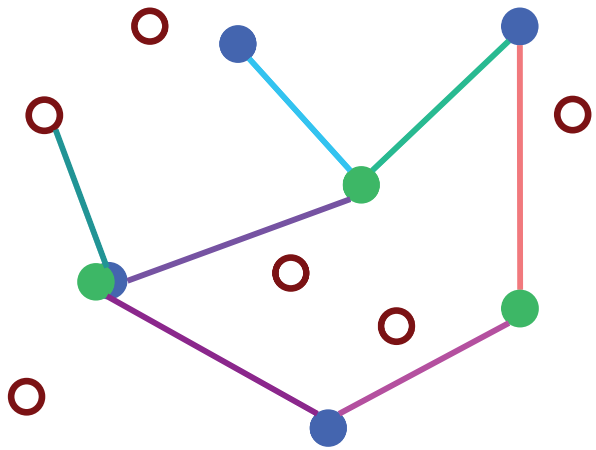
$$H \rightarrow G$$

GreedyMAX

We "shrink" the hypergraph *H* to a graph *G*



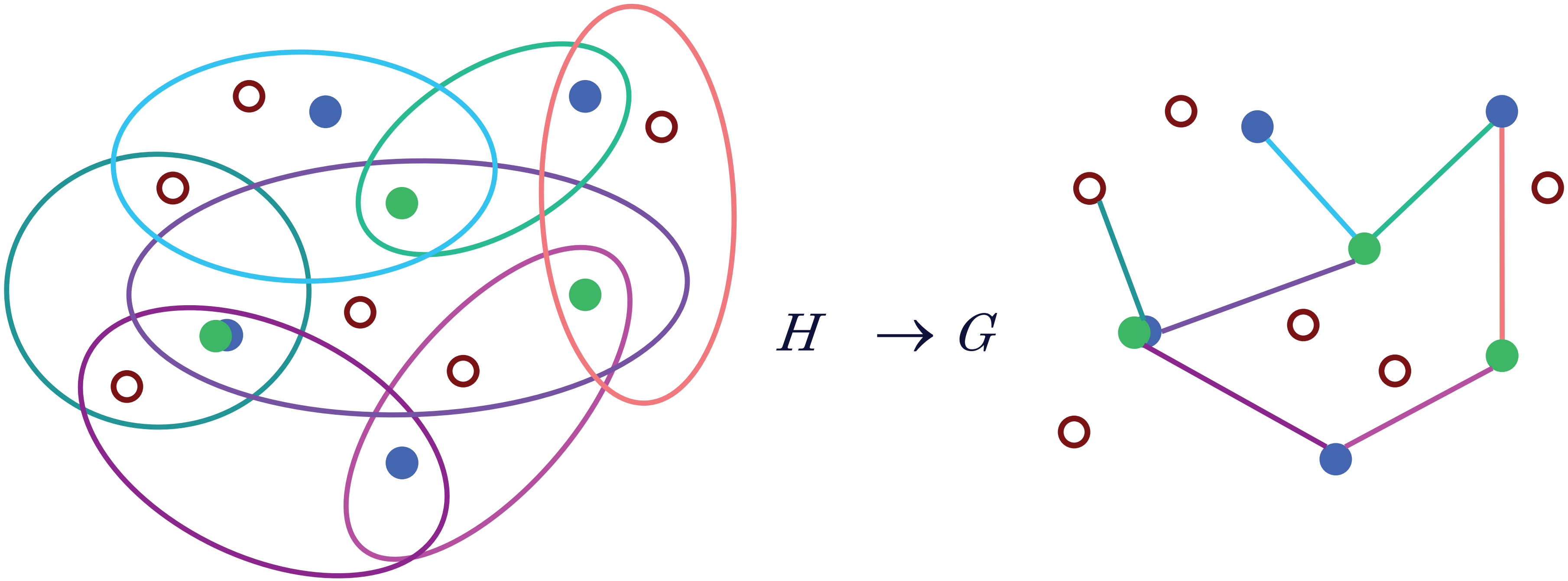
$H \rightarrow G$



V(G) = V(H) and |E(G)| = |E(H)|

Shrinkage properties

1. An optimal cover in G is at most of the same size of an optimal cover in H
2. GreedyMAX constructs the same greedy cover for H and G



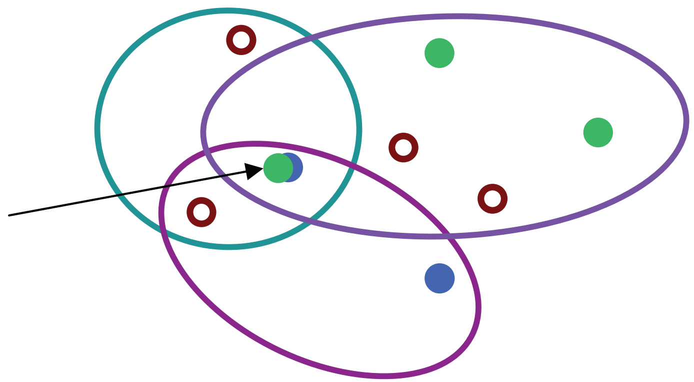
Proof of shrinkage properties

Proof by induction on *s*, the number of iterations of GreedyMAX:

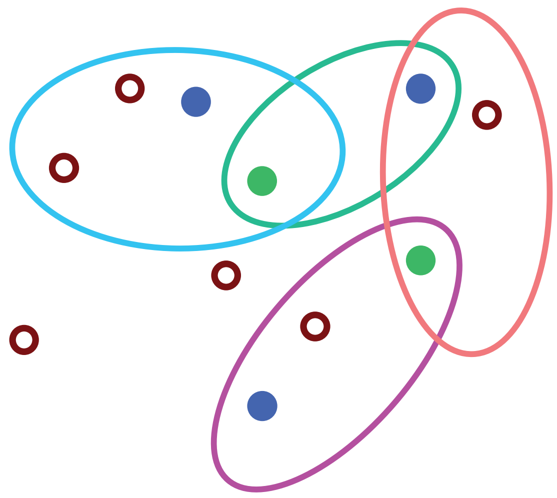
- Base case $s = 0$ trivial.
- Let u_1 = the first vertex chosen by GreedyMAX,
  E_1 = the set of edges incident on u_1
  H_1 = the remaining hypergraph after removing u_1 and E_1

E_1

u_1



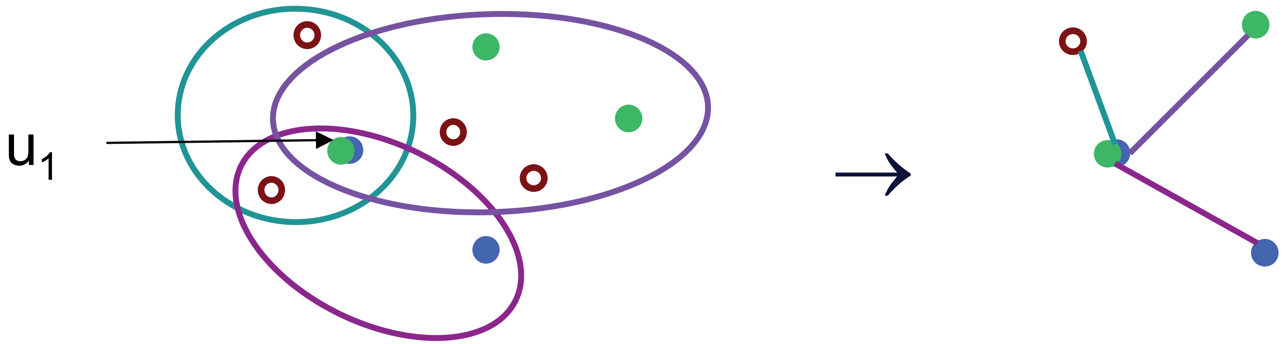
H_1



Rules for shrinking

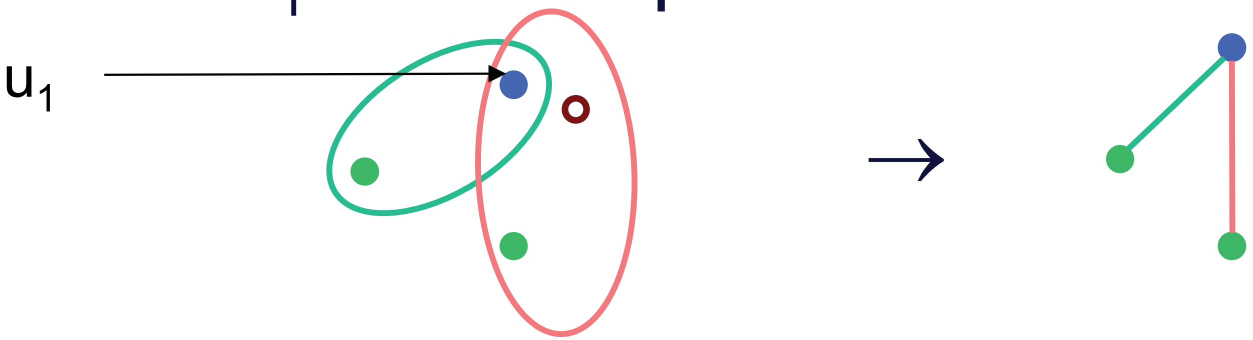
To truncate a hyperedge $e \in E_1$:

- $u_1 \in S^* \cap S$ → pick an *arbitrary* vertex v in *e*



Add u_1v to G

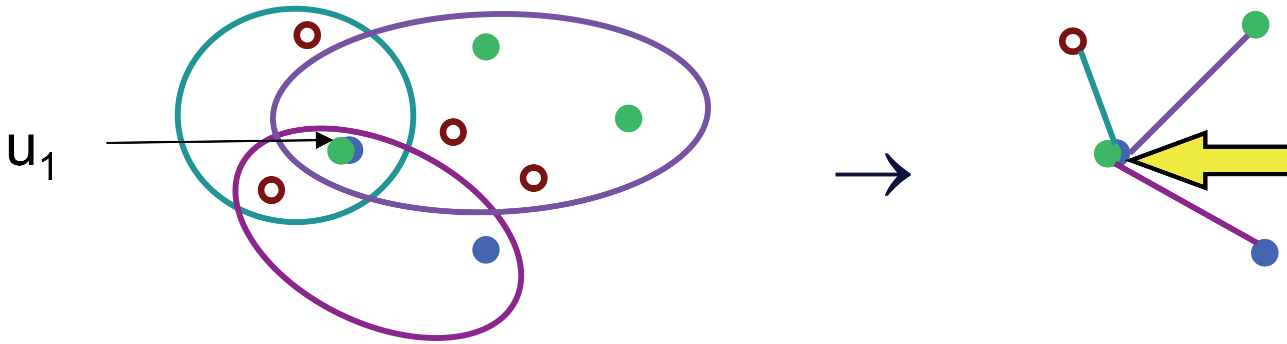
- $u_1 \in S$ → pick a vertex v in *e* such that $v \in S^*$



Rules for shrinking

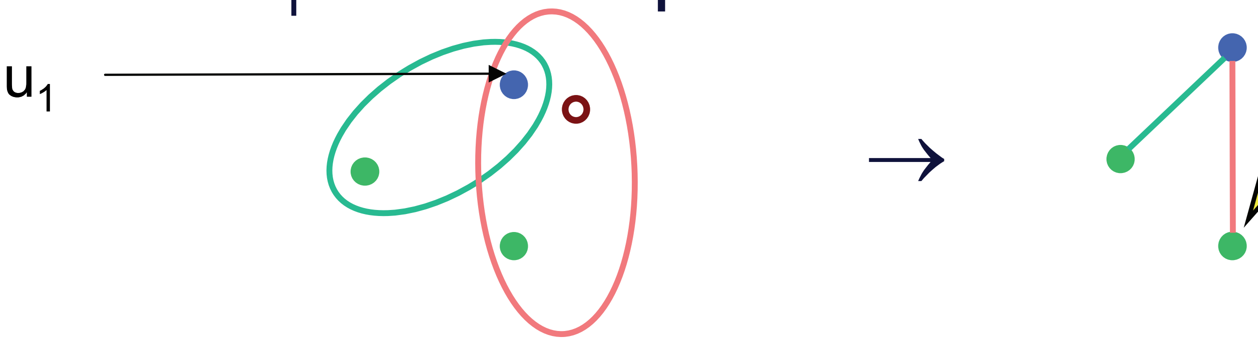
To truncate a hyperedge $e \in E_1$:

- $u_1 \in S^* \cap S \;\rightarrow\;$ pick an *arbitrary* vertex v in *e*



Each edge contains a vertex in S*

- $u_1 \in S \;\rightarrow\;$ pick a vertex v in *e* such that $v \in S^*$

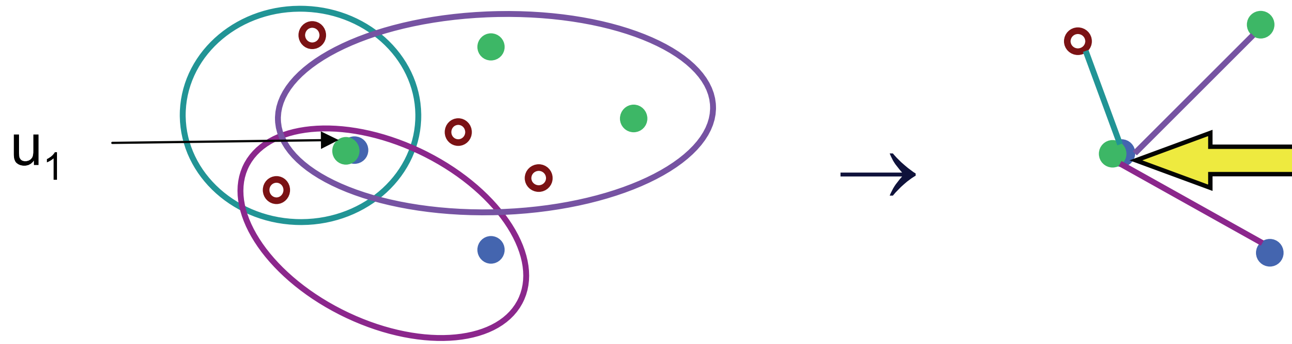


S* is still a cover

Rules for shrinking

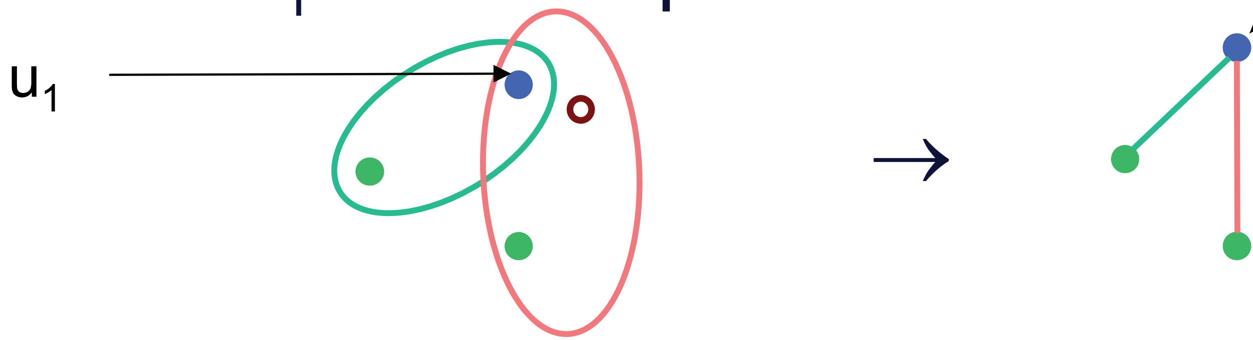
To truncate a hyperedge $e \in E_1$:

- $u_1 \in S^* \cap S \;\rightarrow$ pick an *arbitrary* vertex v in *e*



Degree of u_1 does not change

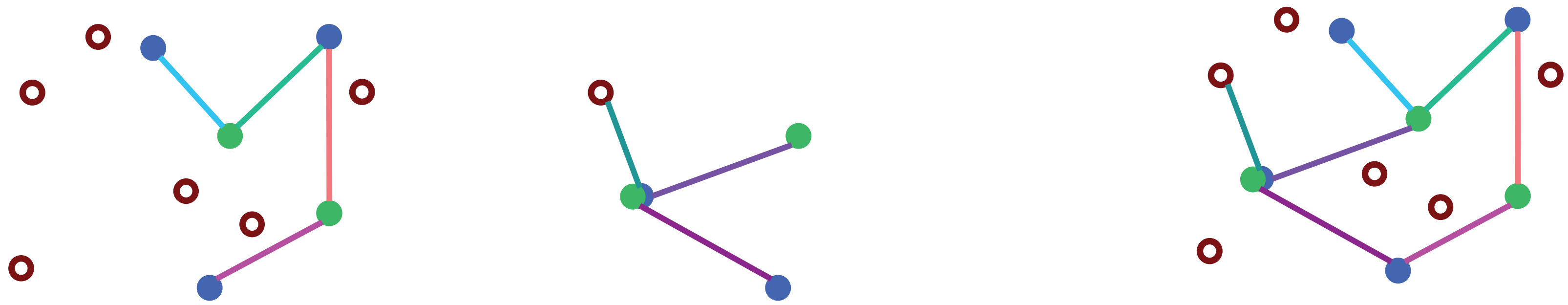
- $u_1 \in S \;\rightarrow$ pick a vertex v in *e* such that $v \in S^*$



Greedy still picks u_1 first

GreedyMAX

- Inductive hypothesis: $G_1 \subseteq H_1$ with greedy cover $S/\{u_1\}$ and is still covered by S^*
- Have: $G \subseteq H$, with $V(G) = V(H)$ and $E(G) = E(G_1) \cup$ truncated E_1



G_1 + E_1 = G

- S^* covers all edges of G $\Rightarrow$ $SC(G) \leq SC(H)$
- The edge shrinkage doesn't decrease the degree of u_1
  $\Rightarrow$ GreedyMAX still selects u_1 first and
  outputs the solution $\{u_1\} \cup S/\{u_1\} = S$

Corollary: Any hypergraph *H* can be shrunk to a graph *G*, for which GreedyMAX has no better performance ratio.

Performance of GreedyMAX

Theorem: GreedyMAX in graphs has ratio $\rho \leq \frac{\Delta+1}{2}$

Proof: We prove a slightly weaker bound.

- An optimal cover satisfies:

  $$|S^*| \geq \frac{m}{\Delta} = \frac{\bar{d}n}{2\Delta}$$

  where *n, m* are the number of vertices and edges, Δ and $\bar{d}$ are the maximum and average degrees

- GreedyMAX attains the Turán bound on graphs [Chvatal,McDiarmid]:

  $$|I| \geq \frac{n}{\bar{d}+1}$$

Performance of GreedyMAX

- Combining

$$|I| \geq \frac{n}{\bar{d}+1} \qquad |S^*| \geq \frac{\bar{d}n}{2\Delta}$$

- The performance ratio is at most

$$\rho = \max_{\forall G} \frac{n-|S^*|}{|I|} \leq \frac{n-\bar{d}/2\Delta}{n/(\bar{d}+1)} = (\bar{d}+1)(1-\bar{d}/2\Delta)$$

- which is maximized when $\bar{d} = \Delta - 1/2$, for the performance ratio $\rho \leq \frac{\Delta+1}{2} + 1/8\Delta$

Tight bounds on GreedyMAX

To get tight bounds, we need two refinements.

We introduce a parameters k $\in$ [0,1] and $d' \leq \Delta - 1$ so that

$$\bar{d} = k\Delta + (1-k)d'$$

Also, we use an extension of the Turan bound, due to Caro & Wei, and proved for GreedyMAX by Sakai,Togasaki,Yamazaki 2003:

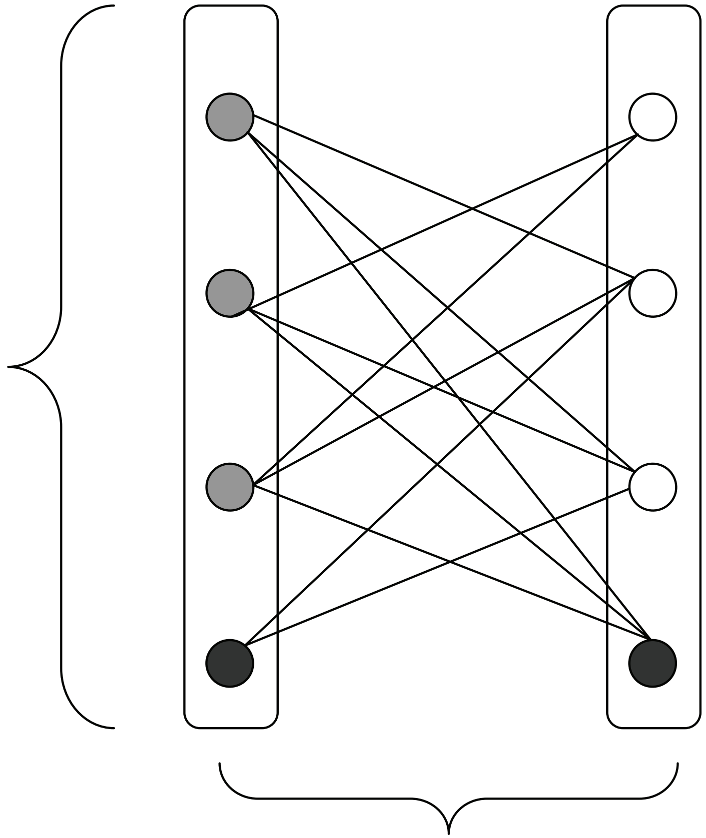
$$|I| \geq \sum_{v \in V} \frac{1}{d(v)+1} = \frac{kn}{\Delta+1} + \frac{(1-k)n}{d'+1}$$

This results in ratio $\rho \leq \frac{\Delta+1}{2}$

Lower bound for GreedyMAX

The performance ratio of GreedyMAXis at least $\frac{\Delta+1}{2}$

OPT = $\Delta + 1$



Greedy =2

Summary

- The performance ratio of GreedyMax for IS in hypergraphs is $(\Delta+1)/2$
  - Obtained by shrinking the hypergraph to a graph, where GreedyMAX does no better
  - Equivalent to differential performance ratio for Set Cover
- One possible lesson: Once you have a proof, find a better proof.

Open problems

- Improve the best known bound of $(\Delta+1)/2$
  - SDP? Gives about $\Delta/\lg \Delta$ ratio for graphs
  - Greediness combined with local search?
- Good lower bounds still missing
- Problem is easier in *k*-uniform hypergraphs
  - $\Delta^{1/(k-1)}$ ratio, obtained by GreedyMAX
  - What other hypergraph properties help?

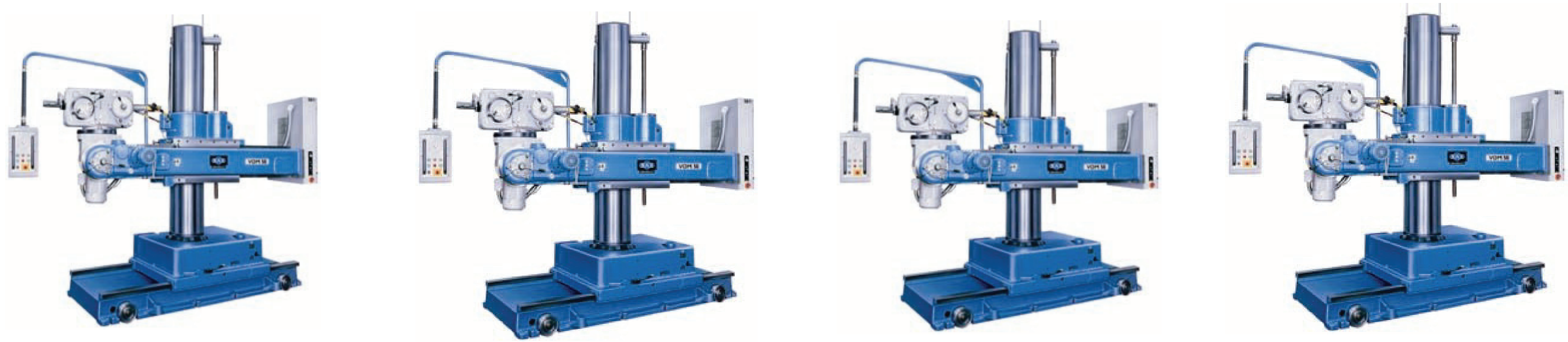
Part IV:
Scheduling with Conflicts

Coloring is a scheduling problem

[Guy Even, H, Lotem Kaplan, Dana Ron 2006]

Scheduling problems

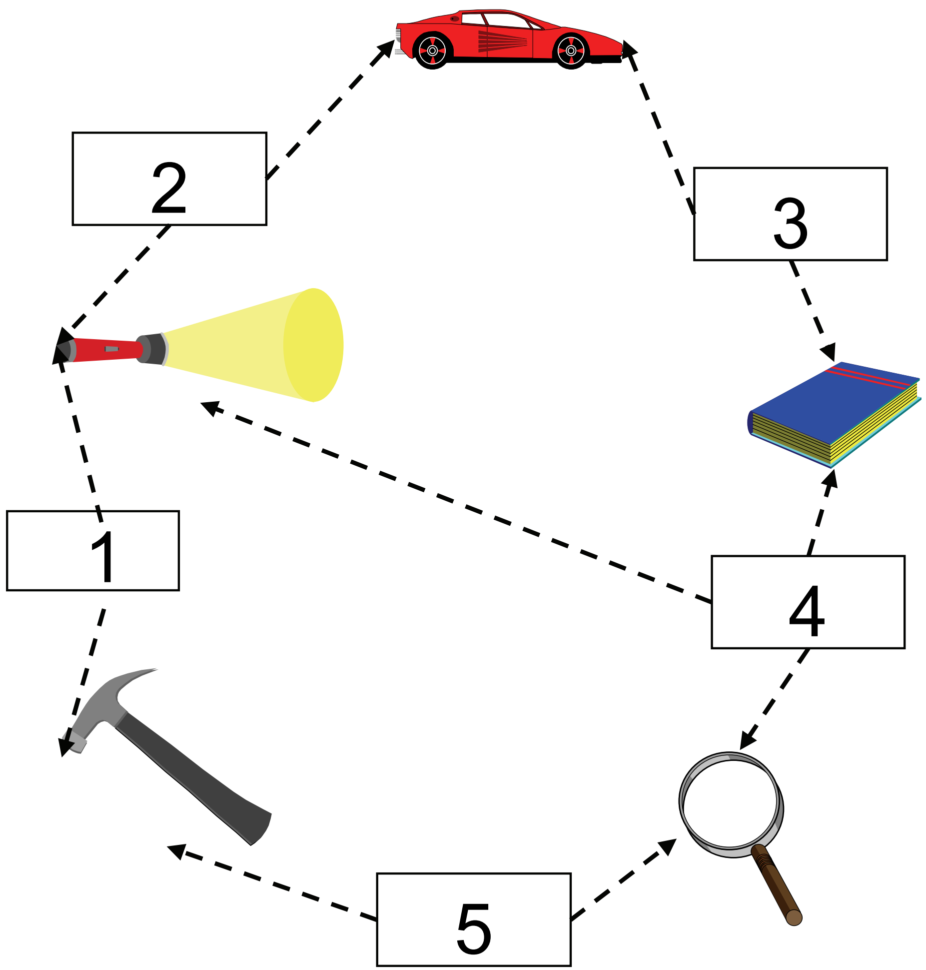
- Given a fixed set of machines
- and a set of jobs to be run on the machines
- Normally, the scheduling problem is an allocation problem, deciding which jobs to allocated to each machine



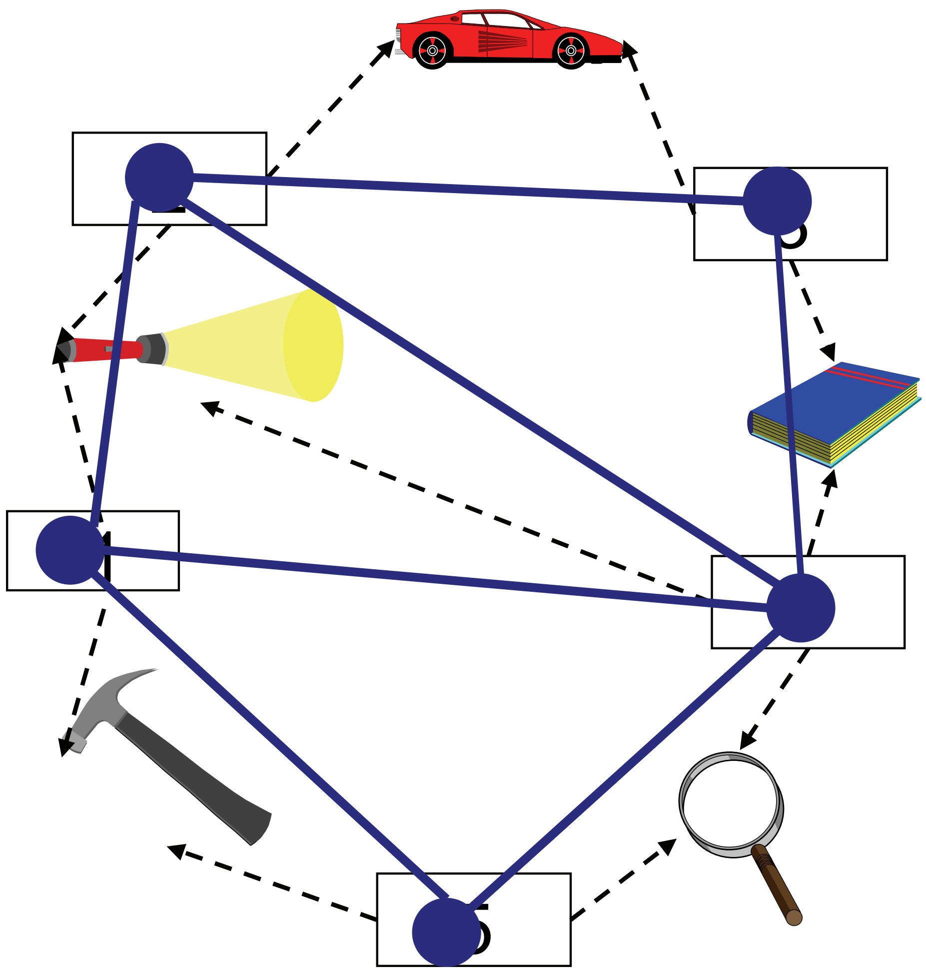
When Coloring meets Scheduling

- Scheduling *dependent* tasks
  - Jobs conflict in that they cannot be executed simultaneously.
- Resource-constrained scheduling
  - Large class of dependent task scheduling
  - Resource:
    - Dedicated processors
    - Bandwidth, (e.g. session scheduling on a LAN)
    - Memory, semiphores, etc.

Resource Constrained Scheduling and Conflict Graph



Resource Constrained Scheduling and Conflict Graph



Main Differences from Coloring

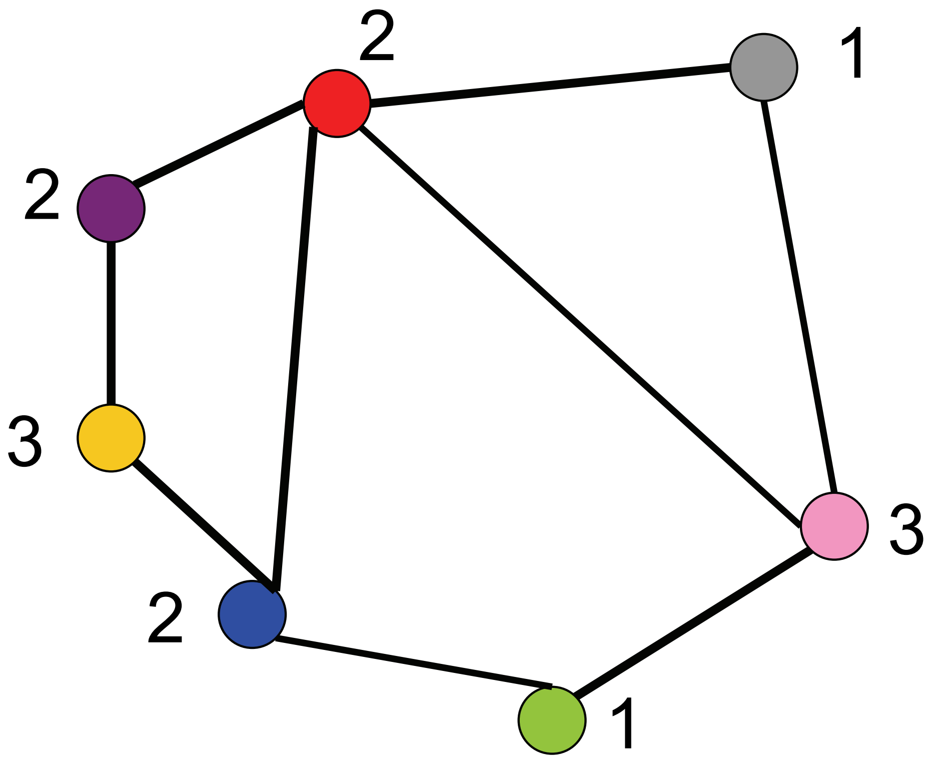
- Correspondence:
  - time step - color
  - job/task - vertex
  - task conflict - edge
- Jobs have ***lengths***
  - Lengths can be different
  - Jobs are run uninterrupted (non-preemptive)
- Fixed number ***m*** of ***machines***
  - At most *m* vertices with each color

Problem Definition

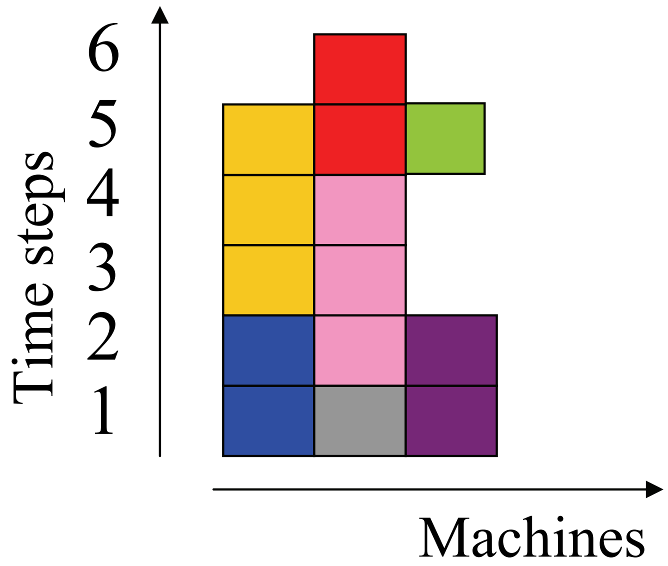
- *Given*: Graph G, and vertex/job lengths p_v
  Number m of machines
- *Find*: A schedule of the jobs so that at *any given time*,
- - at most m jobs are scheduled,
  - no conflicting jobs are scheduled
- *Minimize:* The *makespan* of the schedule, $\max_v x_v + p_v$

Example with m=3 machines

A Conflict Graph



Non-preemptive schedule

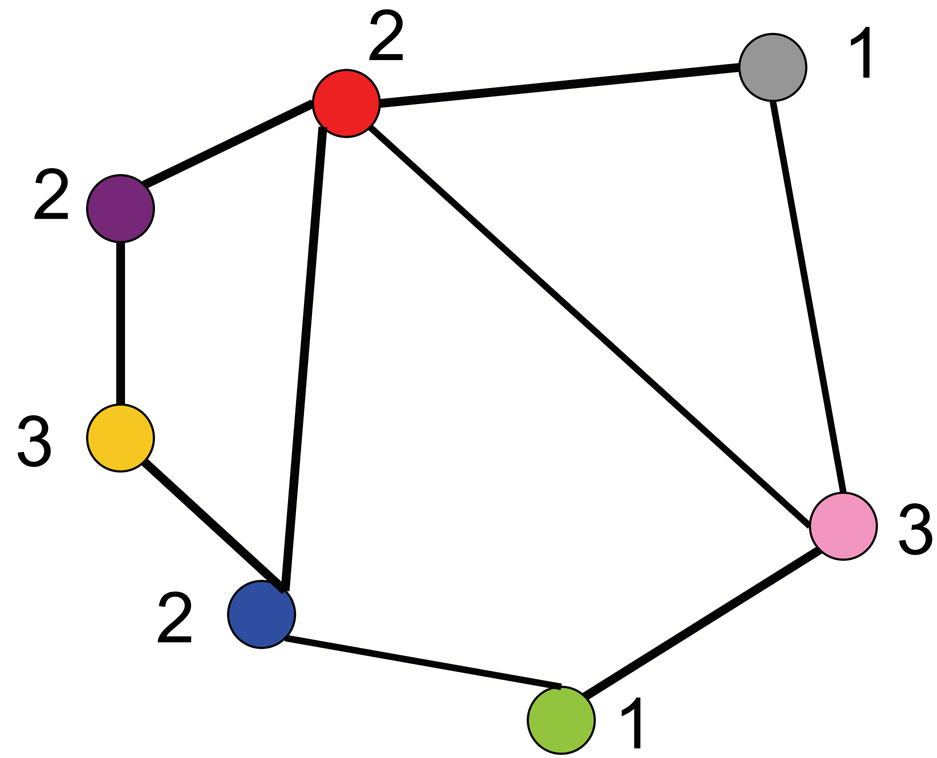


Unit case == Each job of length 1

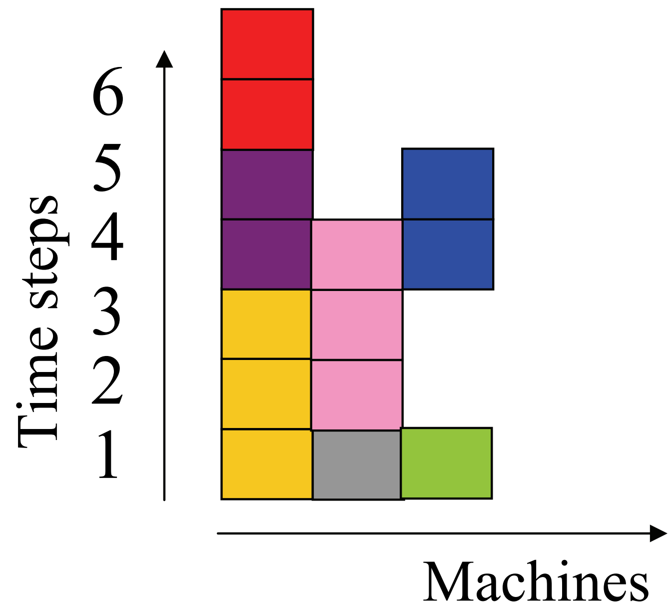
- Equivalent to a version of the k-Set Cover problem
  - Each item v to be covered p_v times
  - Make p_v identical copies of each element (vertex)
  - Each set of size k=m
- Exercise: Show equivalence, assuming p_v constant

Example with m=3 machines

A Conflict Graph



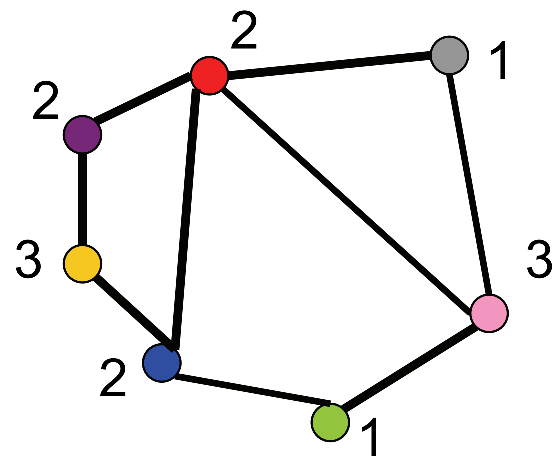
Compact schedule

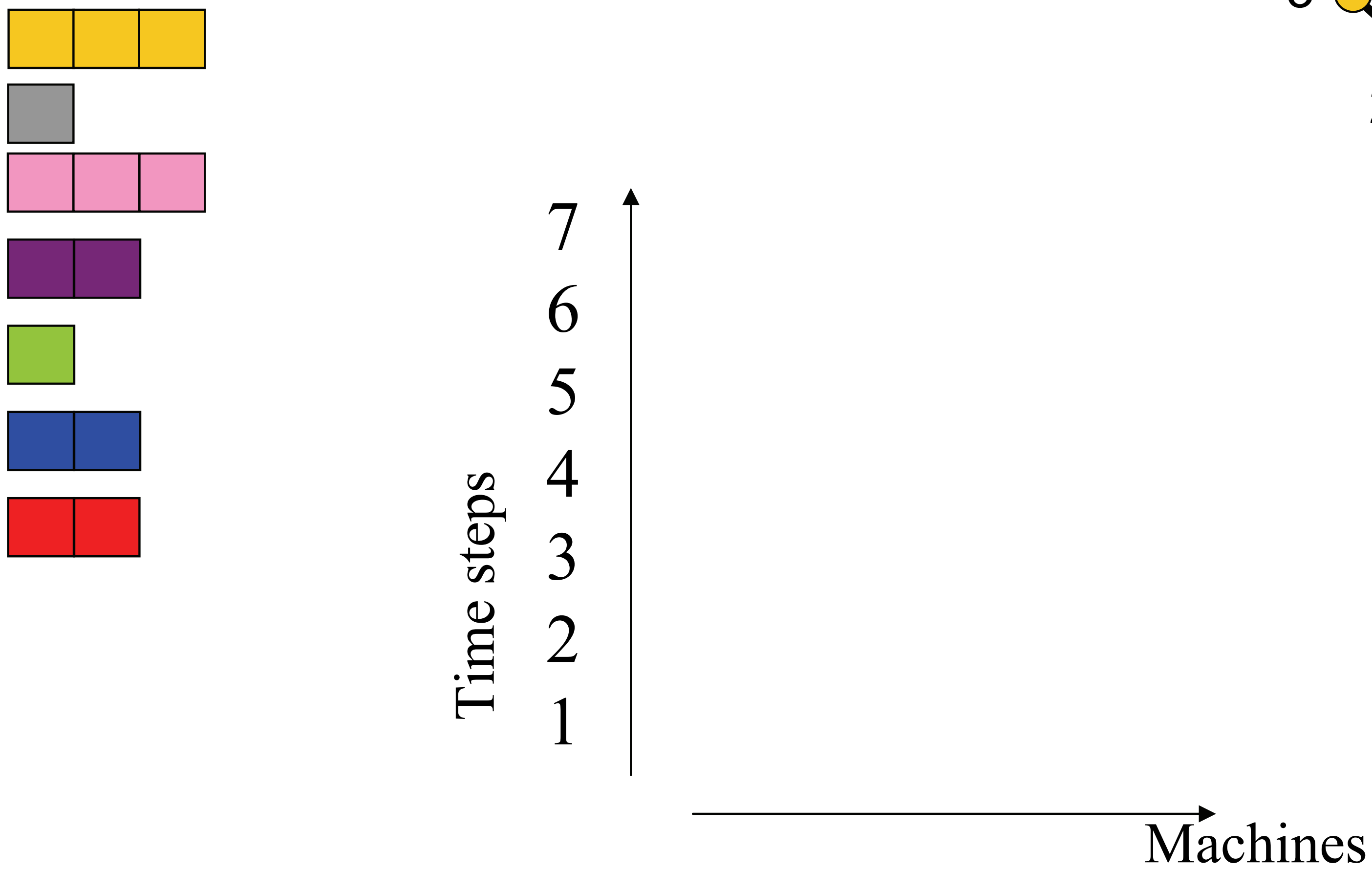


A Greedy Algorithm

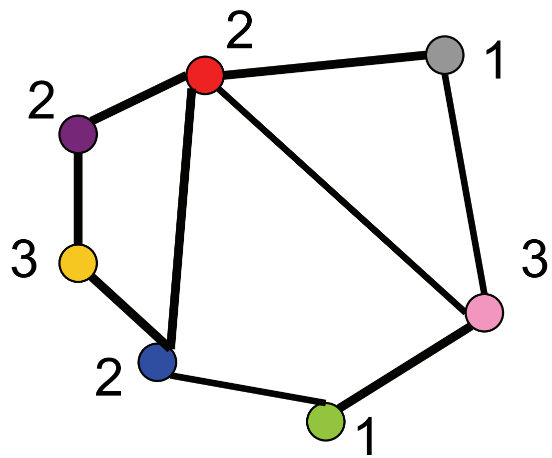
- Among the remaining jobs, pick the one that can be scheduled earliest
  - And fix its schedule

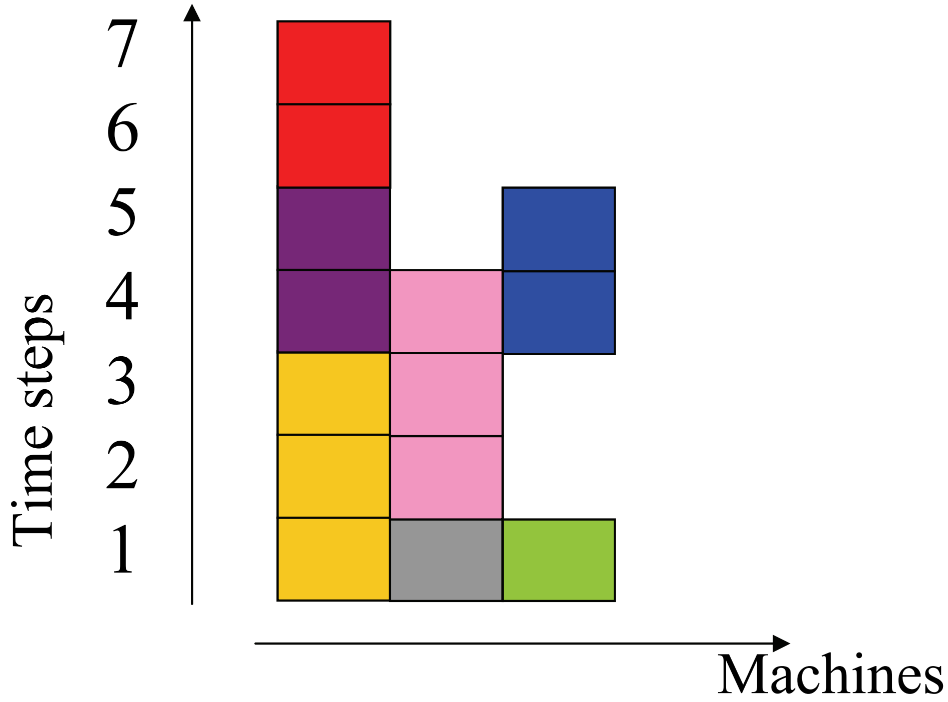
Tetris-like view





Tetris-like view



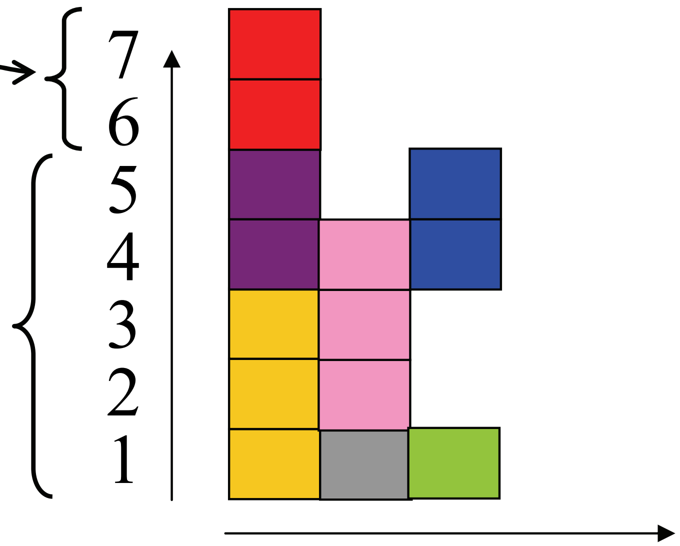


Performance Evaluation

- Any non-trivial algorithm has ratio $\leq m$.
- The greedy coloring achieves a $(m+1)/2$ ratio
  - And this is tight
- Idea of the analysis:
  - Show that in most time steps, the algorithm schedules at least 2 jobs
  - Optimal solution schedules at most *m* jobs in each time step

Analysis

- Count the time steps spent by algorithm
  - ALG = $A_1 + A_2$
  - A_1 = #time steps with only one job scheduled (2)
  - A_2 = #time steps with at least two jobs (5)



OPT = #time steps in optimal solution

Ops = total # operations = $\sum_v x_v$ (13)

$\quad$ OPT $\geq$ Ops / m. (13/3)

Analysis cont.

At least 2 non-A_1 operations in each A_2 step

- Notice:

$$ALG \leq A_1 + (Ops - A_1)/2 \leq (A_1 + Ops)/2$$

Need to show that A_1 is not too big

Analysis cont.

- Claim: Any two jobs in A_1 must conflict
  - After one of them was fixed, the other was one NOT scheduled alongside the first one
- Thus, $OPT \geq A_1$
- Conclusion:

$$ALG \leq (A_1 + Ops)/2 \leq (OPT + m*OPT)/2$$

OPT performs at most *m* operations in each step

Open questions:

- Improve the (m+1)/2-approximation
- Is there a (nearly) linear lower bound?
  - Applies to many multicoloring questions

Part V: Bounded-degree graphs

Simple partitioning

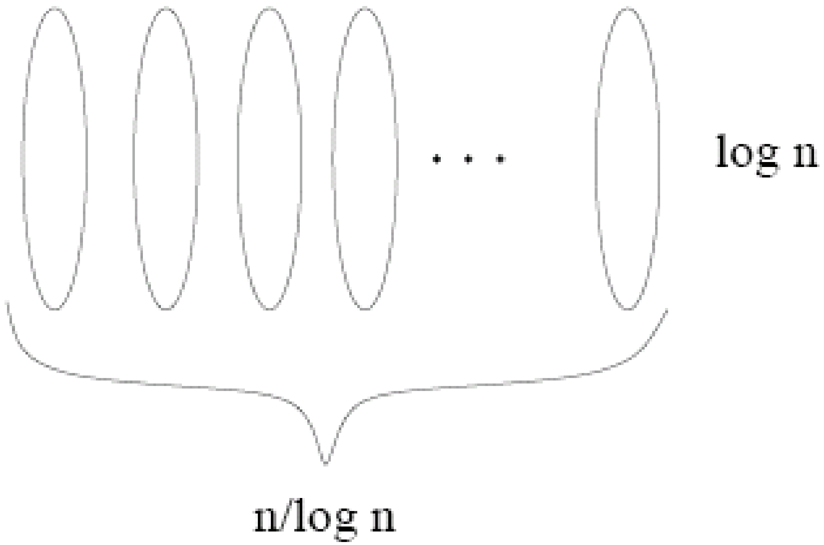
[H, Lau, 1995]

Coloring of bounded-degree graphs

- Simple algorithm gives $\lceil(\Delta+1)/4\rceil$ ratio
  - Partition graph into subgraphs of degree 3
  - Solve each subgraph optimally
- Asymptotically better algorithm using SDP
  - Semi-definite programming
  - $O(\Delta^{1/(\chi-1)} \log n)$ ratio

Coloring General Graphs

- Break the graph into $n/\log n$ sets of $\log n$ vertices each.
- Solve each subgraph *exhaustively*.



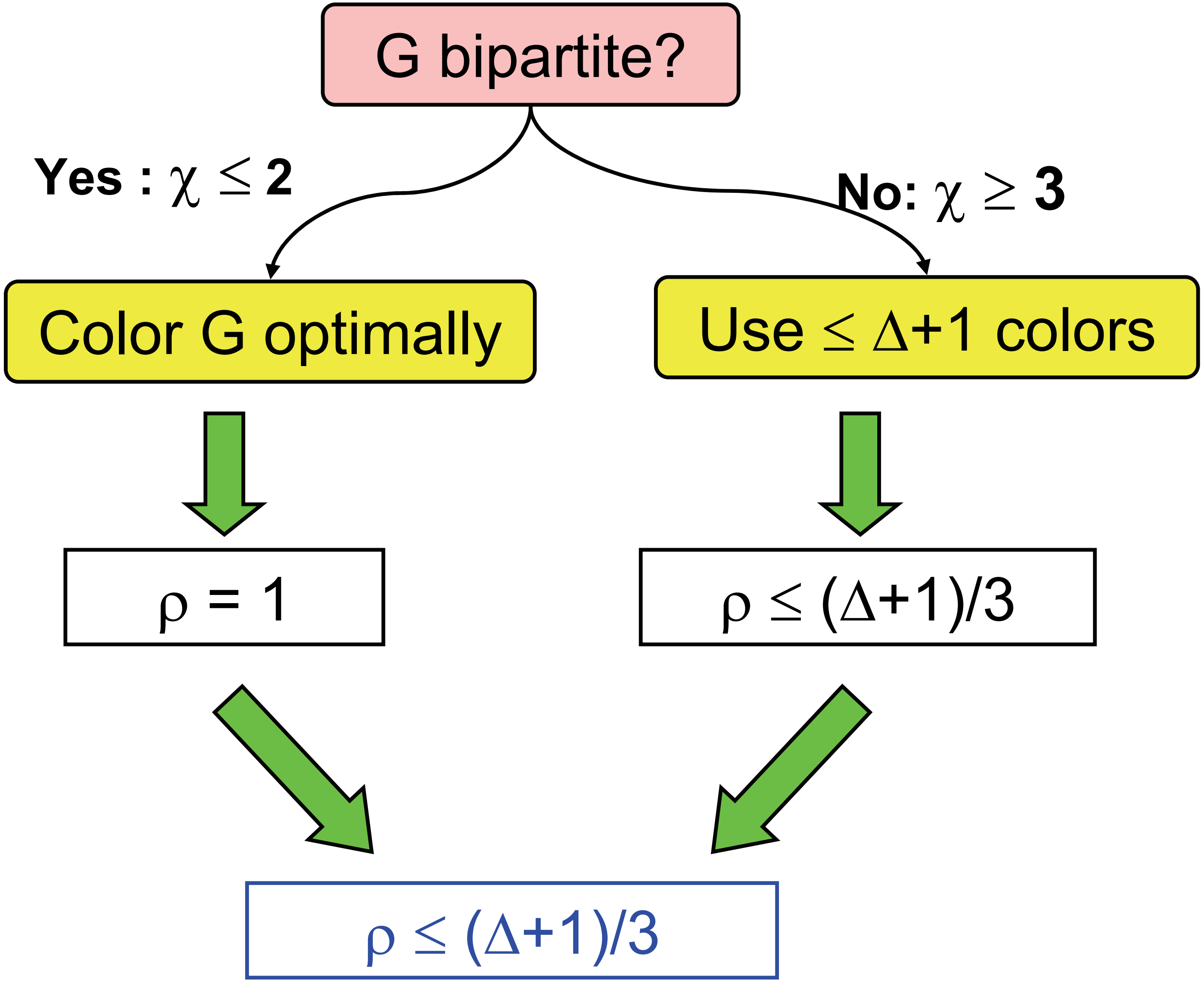
Breaking a graph into $n/\log n$ *sets of size* $\log n$ *each*

Complexity:

$$n/\log n \cdot 2^{\log n} \cdot \log^2 n = n^2 \log n.$$

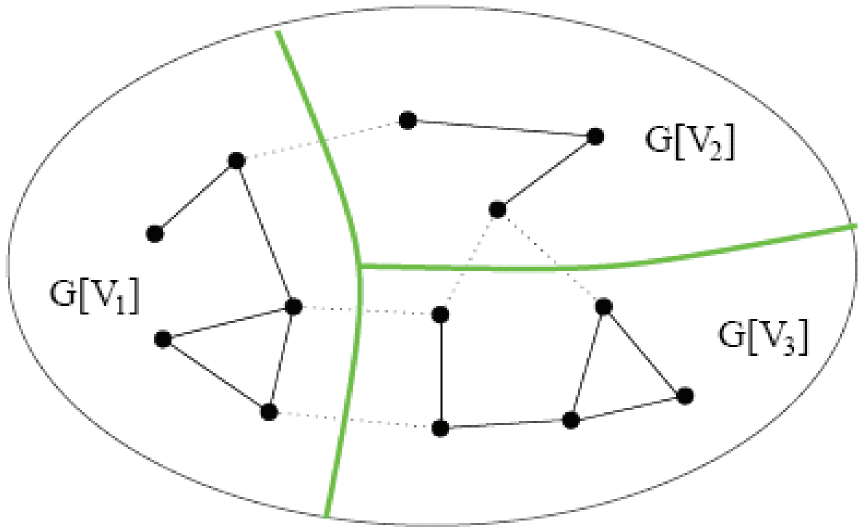
Result: $n/\log n$-approximation ratio

Easy coloring of bounded-degree graphs



Simple Partitioning Argument

- Suppose we break a graph (partition the vertices) into *t* parts, and solve each part optimally.
- Then, the combined solution is a *t*-approximation for coloring the original graph



Lovász' Partitioning Lemma

Let us first state the lemma more generally.

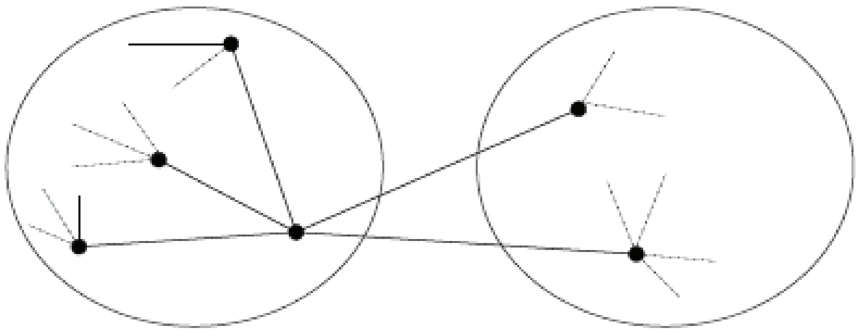
Lemma 4 (Lovász) *Let $G = (V, E)$ be graph of maximum degree Δ, and let k be a positive integer. Then, V can be partitioned into $t = \lceil (\Delta + 1)/(k + 1) \rceil$ subsets $V_1, \ldots, V_t$, such that $\Delta(G[V_i]) \leq k$ for $i = 1, 2, \ldots t$.*

Local search algorithm:

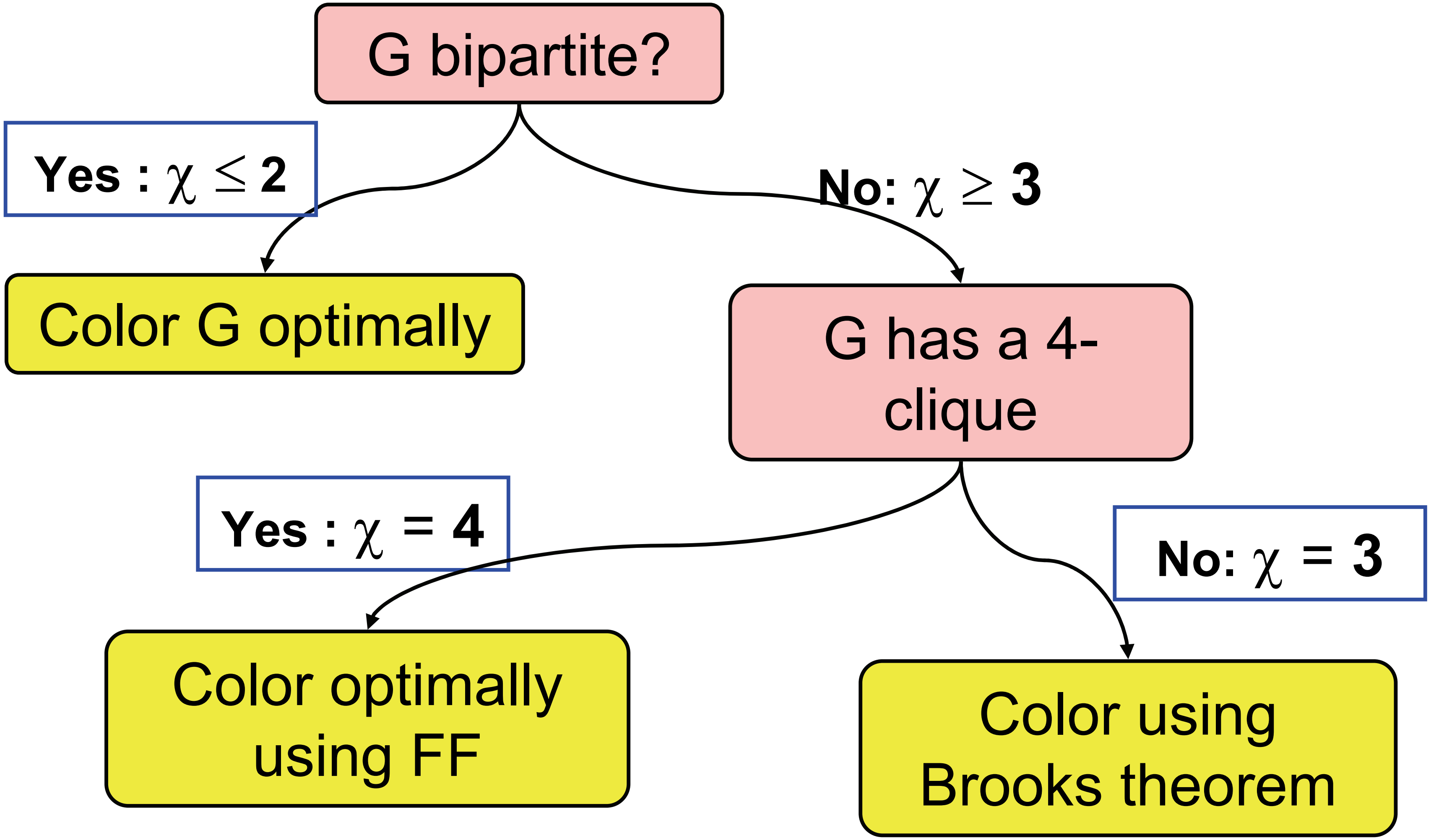
Start with an arbitrary partition. If there is a vertex v of degree more than k in the current subgraph, move it to a subgraph where it has k or fewer neighbors. Repeat the above operation as often as needed.

Observe: $t \cdot (k + 1) > \Delta$, so v cannot have $k + 1$ neighbors in every subgraph.

Potential function: The number of *edges* crossing subgraphs in the partition.



Exact coloring of degree-3 graphs



Corollary:

- Linear time $\lceil(\Delta+1)/4\rceil$ approximation
  - Can be reduced to $(\Delta+3)/4$