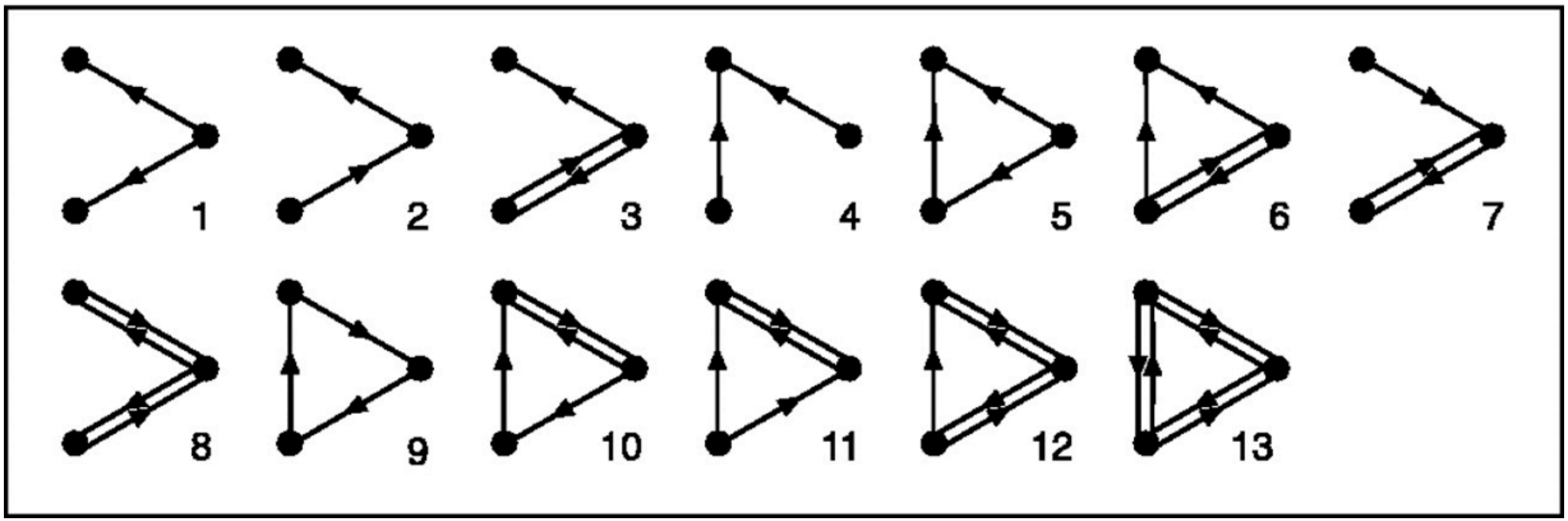
Network Motifs: Simple Building Blocks of Complex Networks

Milo et al., Science, 2002.

Beyond Degree Distribution & Diameter

Network Motifs: Consider all possible ways to connect 3 nodes with directed edges:



(Milo et al., *Science*, 2002)

Finding Over-represented Subgraphs

For each possible motif M:

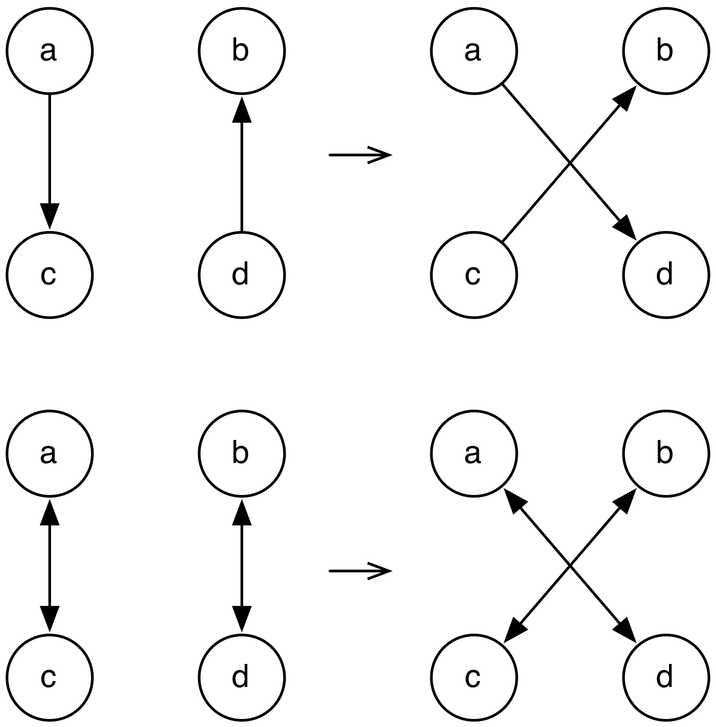
Let c_M be the number of times M occurs in graph G.

Estimate $p_M = \Pr[\#\text{ occurrences} \geq c_M]$ when edges are shuffled.

Output M if $p_M < 0.01$ and $c_M > 4$.

To generate a random graph for the 3-node motifs:

Single and double edges swapped separately:



To Generate Random Graphs With a Given Distribution of (n-1)-node subgraphs:

Define an "energy" on a vector of occurrences of motifs:

$$\text{Energy}(V_{\text{rand}}) = \sum_M \frac{|V_{\text{real},M} - V_{\text{rand},M}|}{(V_{\text{real},M} + V_{\text{rand},M})}$$

When $V_{\text{rand}} = V_{\text{real}}$, the energy is 0.

Start with a randomized network.
Until Energy is small:
    Make a random swap.
    If the swap reduces the energy, keep it
    Otherwise, keep it with probability $\exp(-\Delta E/T)$

Network	Nodes	Edges	N_{real}	$N_{rand} \pm$ SD	Z score	N_{real}	$N_{rand} \pm$ SD	Z score	N_{real}	$N_{rand} \pm$ SD	Z score
Gene regulation (transcription)					**Feed-forward loop**			**Bi-fan**			
E. coli	424	519	40	7 ± 3	10	203	47 ± 12	13			
*S. cerevisiae**	685	1,052	70	11 ± 4	14	1812	300 ± 40	41			
Neurons					**Feed-forward loop**			**Bi-fan**			**Bi-parallel**
C. elegans†	252	509	125	90 ± 10	3.7	127	55 ± 13	5.3	227	35 ± 10	20
Food webs					**Three chain**			**Bi-parallel**			
Little Rock	92	984	3219	3120 ± 50	2.1	7295	2220 ± 210	25			
Ythan	83	391	1182	1020 ± 20	7.2	1357	230 ± 50	23			
St. Martin	42	205	469	450 ± 10	NS	382	130 ± 20	12			
Chesapeake	31	67	80	82 ± 4	NS	26	5 ± 2	8			
Coachella	29	243	279	235 ± 12	3.6	181	80 ± 20	5			
Skipwith	25	189	184	150 ± 7	5.5	397	80 ± 25	13			
B. Brook	25	104	181	130 ± 7	7.4	267	30 ± 7	32			
Electronic circuits (forward logic chips)					**Feed-forward loop**			**Bi-fan**			**Bi-parallel**
s15850	10,383	14,240	424	2 ± 2	285	1040	1 ± 1	1200	480	2 ± 1	335
s38584	20,717	34,204	413	10 ± 3	120	1739	6 ± 2	800	711	9 ± 2	320
s38417	23,843	33,661	612	3 ± 2	400	2404	1 ± 1	2550	531	2 ± 2	340
s9234	5,844	8,197	211	2 ± 1	140	754	1 ± 1	1050	209	1 ± 1	200
s13207	8,651	11,831	403	2 ± 1	225	4445	1 ± 1	4950	264	2 ± 1	200
Electronic circuits (digital fractional multipliers)					**Three-node feedback loop**			**Bi-fan**			**Four-node feedback loop**
s208	122	189	10	1 ± 1	9	4	1 ± 1	3.8	5	1 ± 1	5
s420	252	399	20	1 ± 1	18	10	1 ± 1	10	11	1 ± 1	11
s838‡	512	819	40	1 ± 1	38	22	1 ± 1	20	23	1 ± 1	25
World Wide Web					**Feedback with two mutual dyads**			**Fully connected triad**			**Uplinked mutual dyad**
nd.edu§	325,729	1.46e6	1.1e5	2e3 ± 1e2	800	6.8e6	5e4±4e2	15,000	1.2e6	1e4 ± 2e2	5000

"Information processing" networks tend to use the same motifs

Other networks each had their own distinct collection of motifs.

Feed forward, e.g.: filter out transient signals.

(Milo et al., *Science*, 2002)

Quickly Finding Motifs

858L

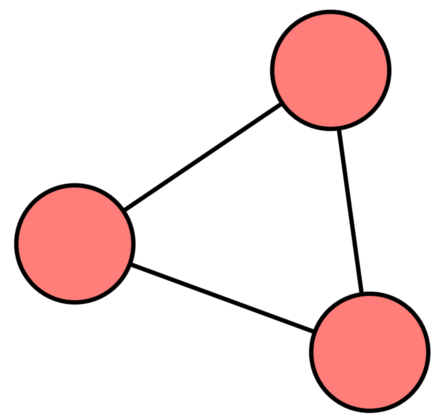
Network Motif Discovery Using Subgraph Enumeration and Symmetry-Breaking

Grochow & Kellis, RECOMB 2007

Backtracking (Recursive) Algorithm to Find Network Motifs

H =

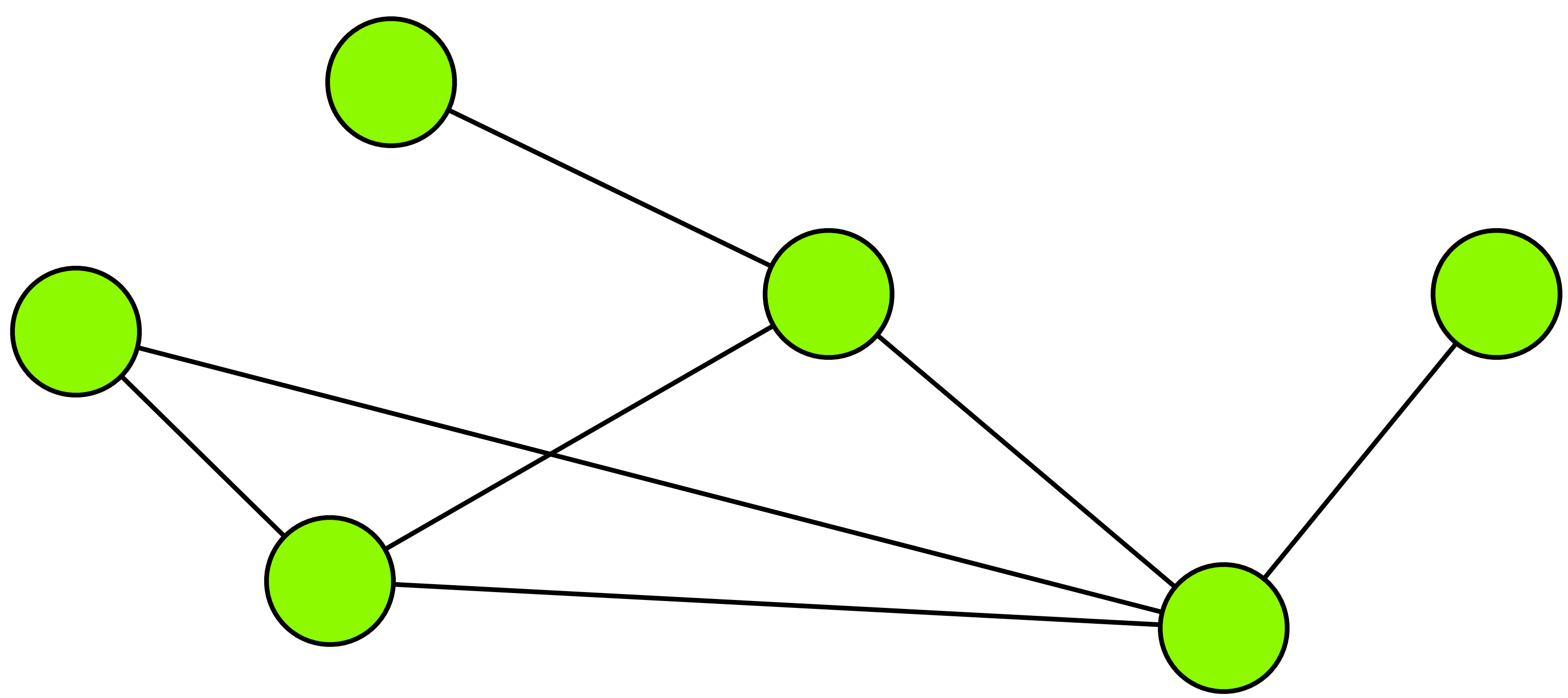
(Small query graph)



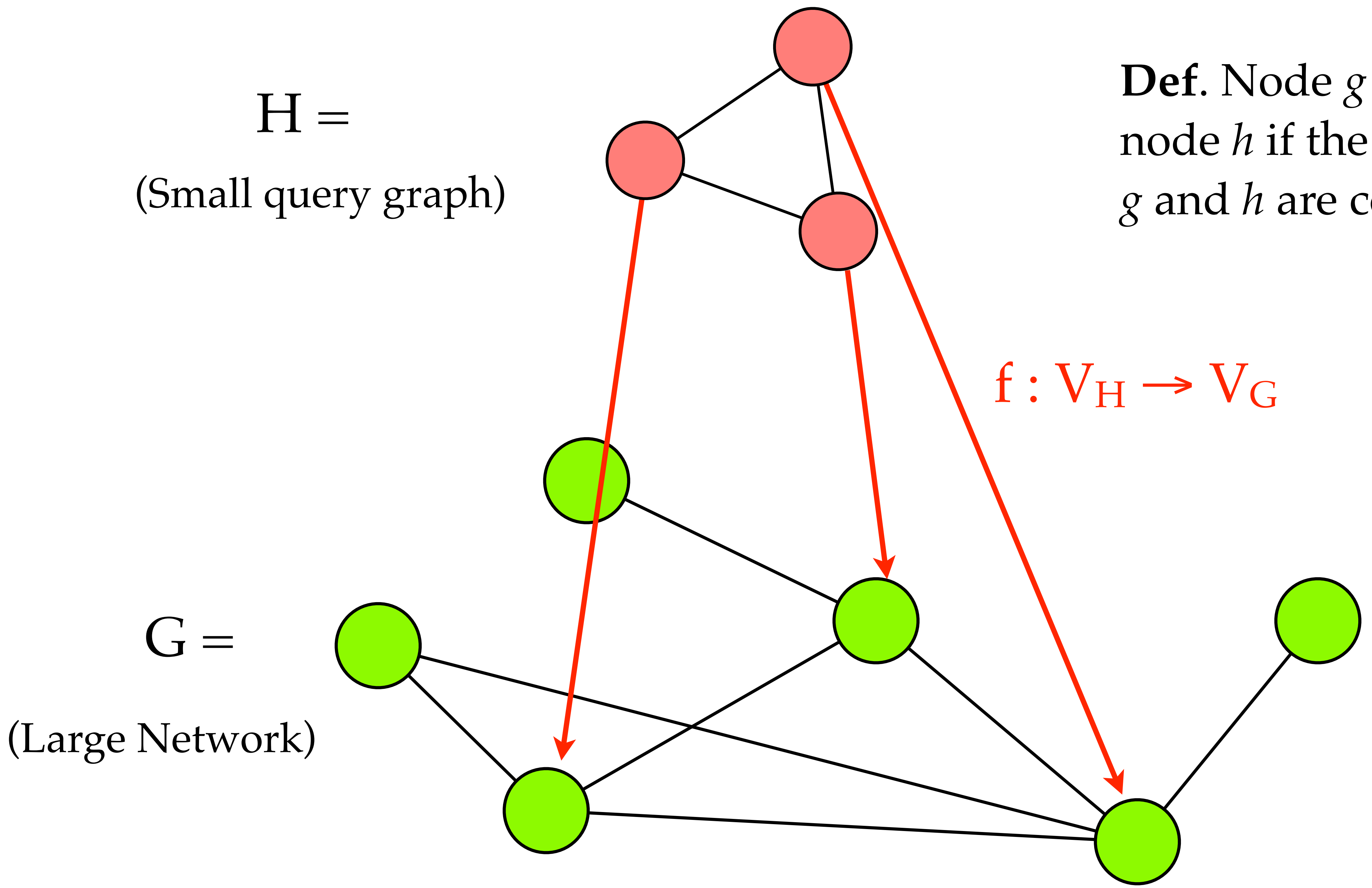
Def. Node *g* **supports** node *h* if the degrees of *g* and *h* are compatible.

G =

(Large Network)



Backtracking (Recursive) Algorithm to Find Network Motifs



Def. Node *g* **supports** node *h* if the degrees of *g* and *h* are compatible.

Basic Algorithm:

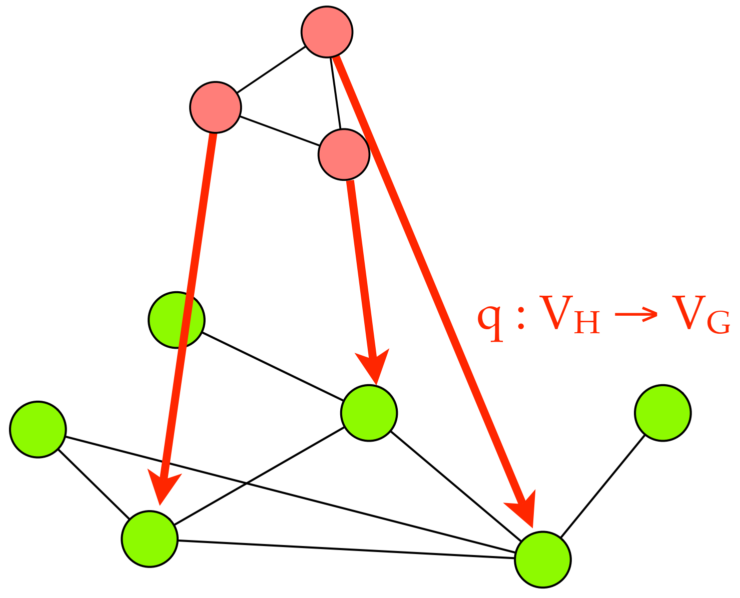
```
For each node g ∈ G
  For each node h ∈ H
    If h can't support g: continue

    Let f = {(g→h)}
    L = Extend(f, G, H)
    For q in L:
      Output image of q
Remove g from G
```

For every possible mapping of a single node from G to H

f is a partial map that maps g to h.

Then grow this partial map into many full maps

No need to consider g again (since we tried all its possible matches already)



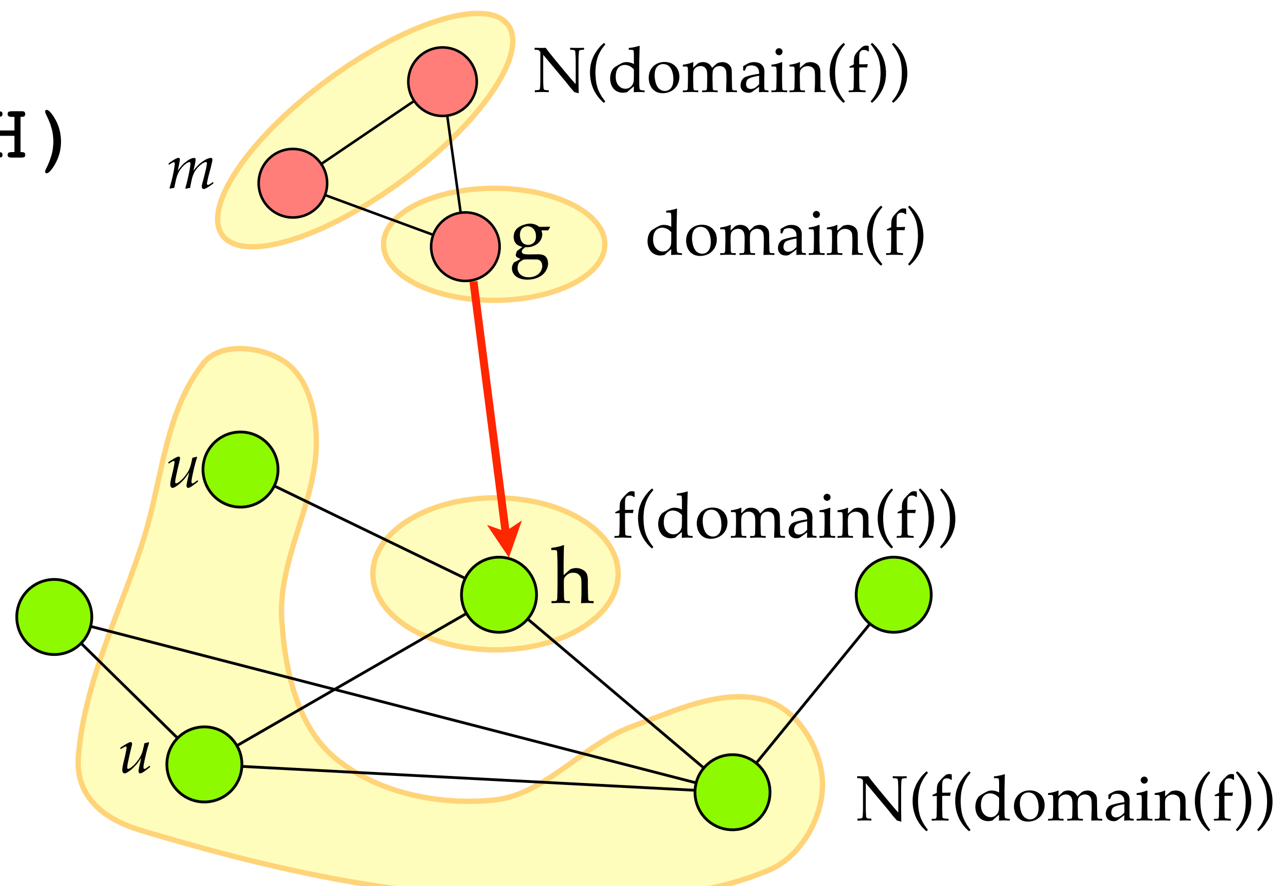
Extend(f, G, H):

```
If domain(f) = H: return [f]

Let m = some node in N(domain(f))
For each node u ∈ N(f(domain(f))):

  If adding (m→u) to f keeps f as a
  valid isomorphism then:
    Extend(f∪{(m→u)}, G, H)
```

Base case

Choose a node in H

Try to map it to G



Extend(f, G, H):

```
If domain(f) = H: return [f]
```

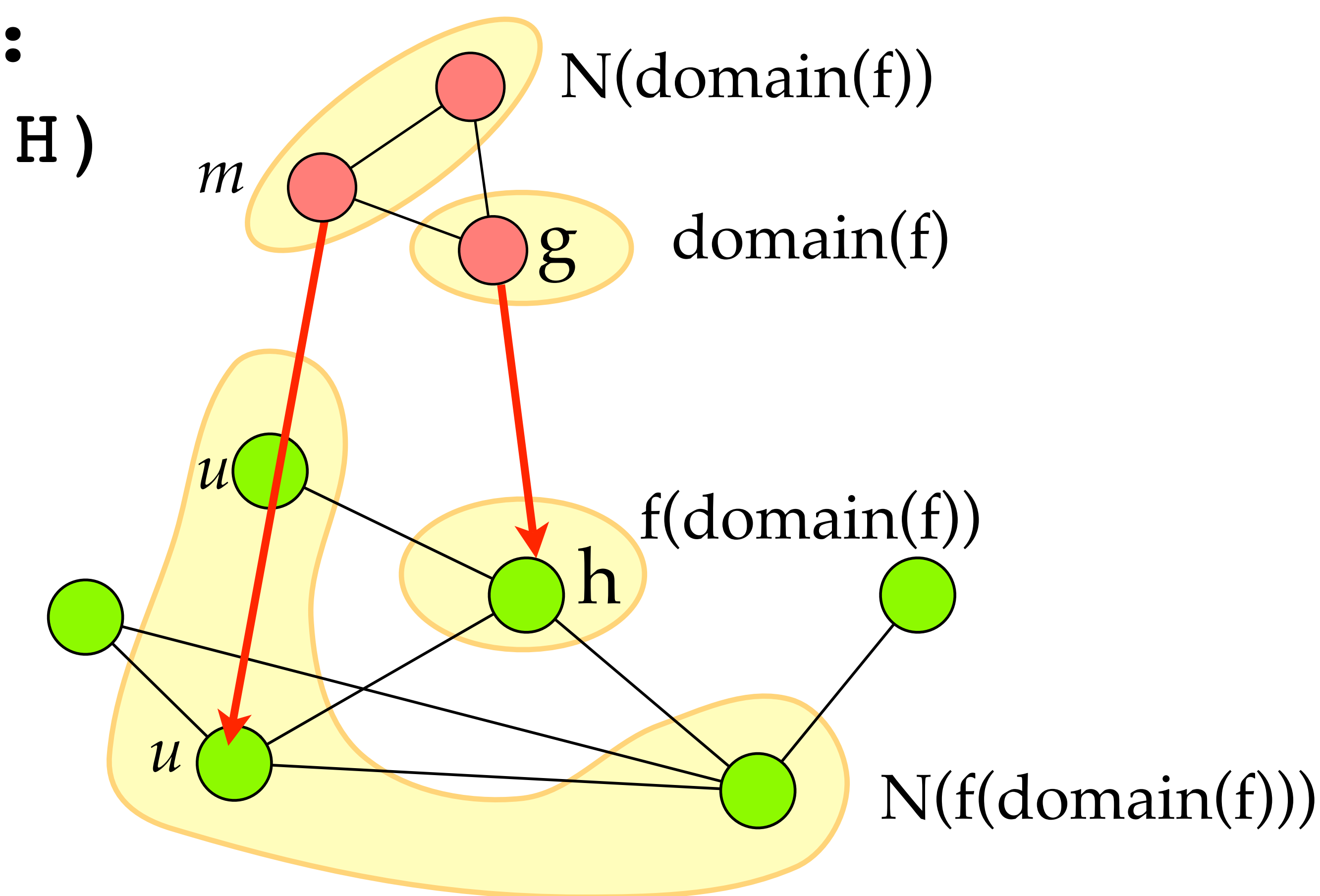
Base case

```
Let m = some node in N(domain(f))
For each node u ∊ N(f(domain(f))):
```

Choose a node in H

Try to map it to G

```
  If adding (m→u) to f keeps f as a
  valid isomorphism then:
    Extend(f∪{(m→u)}, G, H)
```



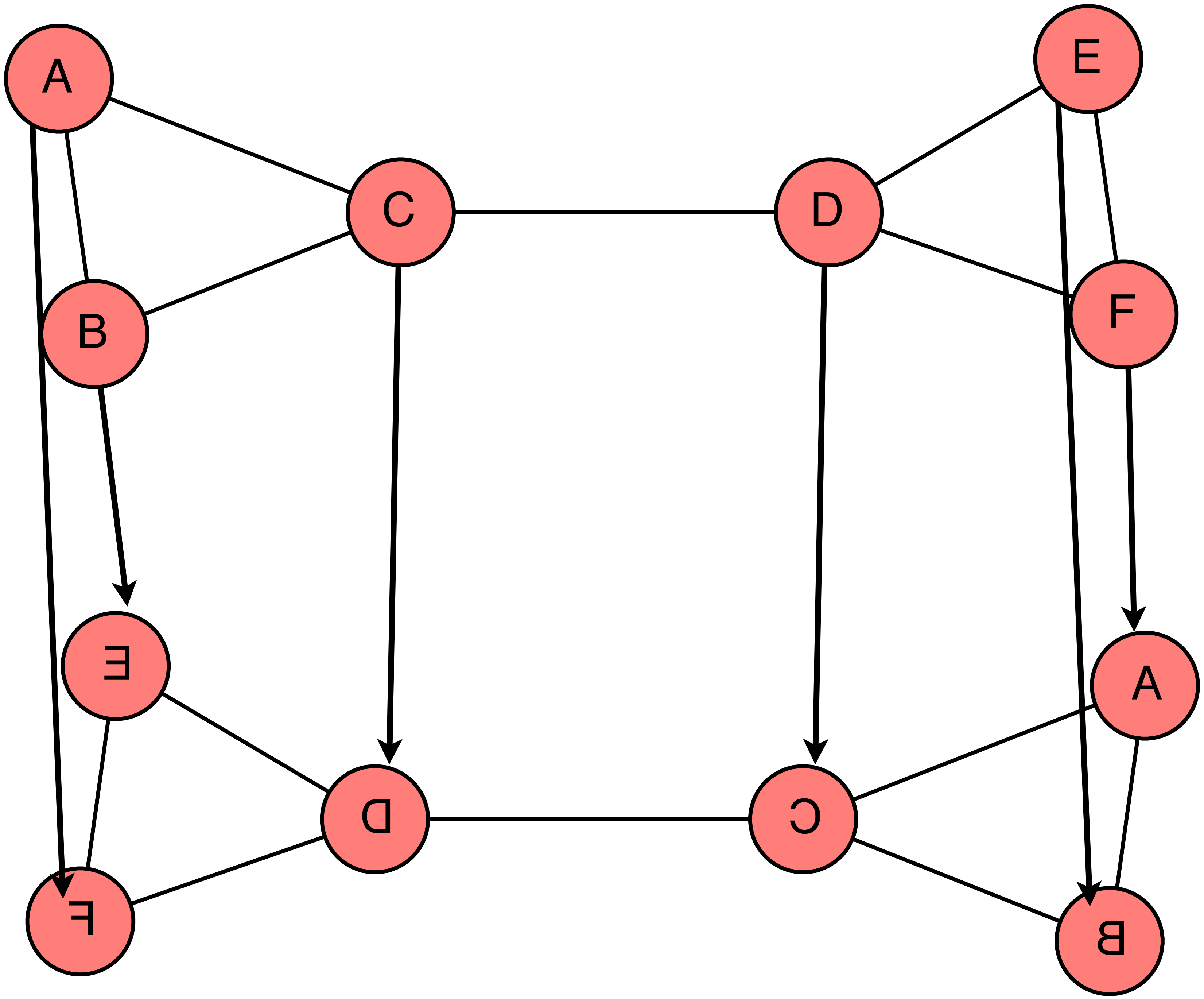
Speed-up #1

- Every time we can choose a node, we pick the one that is "most constrained":
  - Pick the node that already has the most mapped neighbors
  - If there are ties, choose the node with the highest degree
  - If there are still ties, choose the node with highest 2nd order degree (total degree of the neighbors)

- Just a heuristic --- doesn't hurt because we can pick the nodes in any order we want
  - if a map that we are building can't be completed, we want to know sooner rather than later.

Automorphisms & Orbits

Def. An **automorphism** is an isomorphism from a graph to itself.

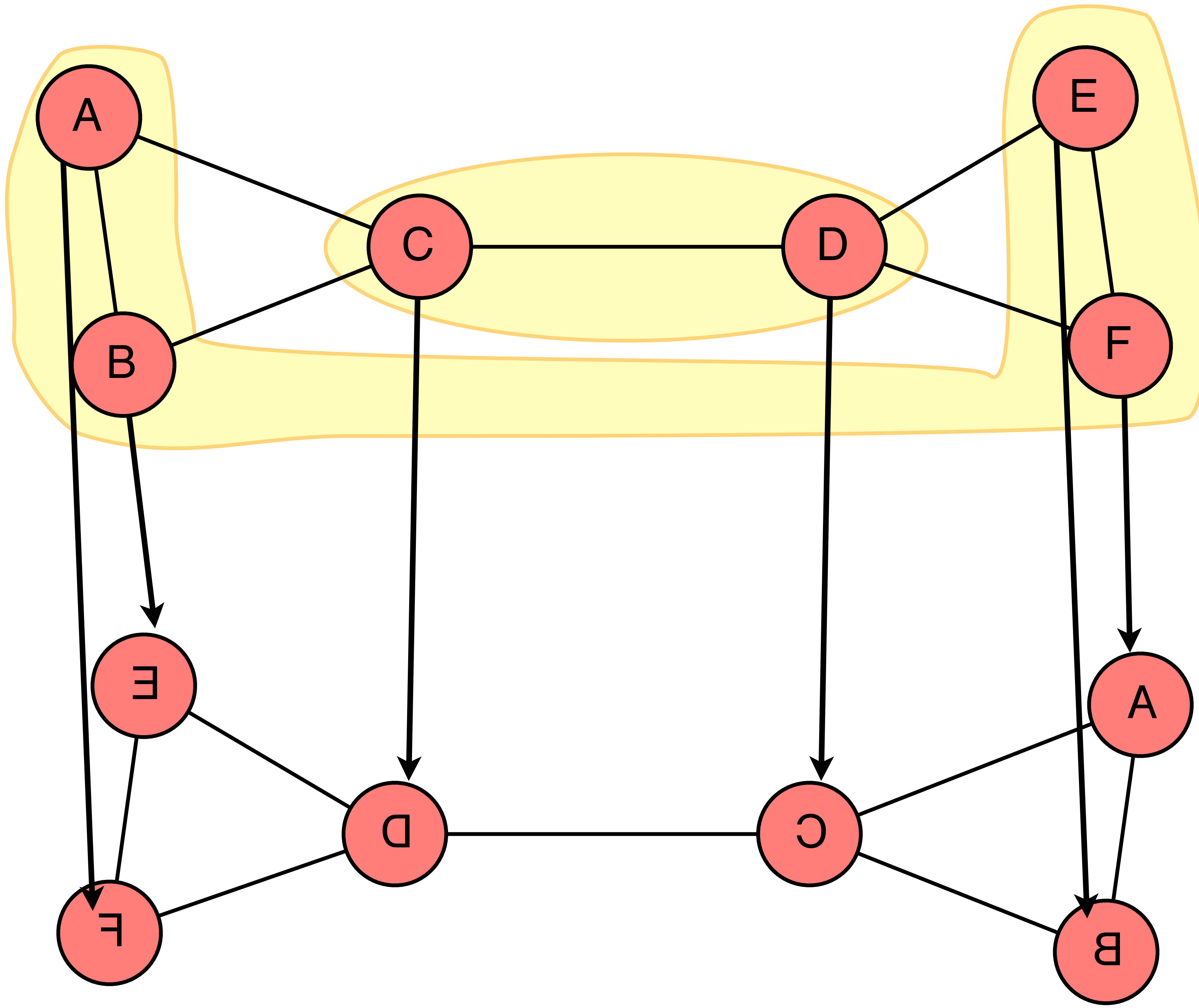


Orbit of a node *u* is the set of nodes that *u* is mapped to under some automorphism

Automorphisms & Orbits

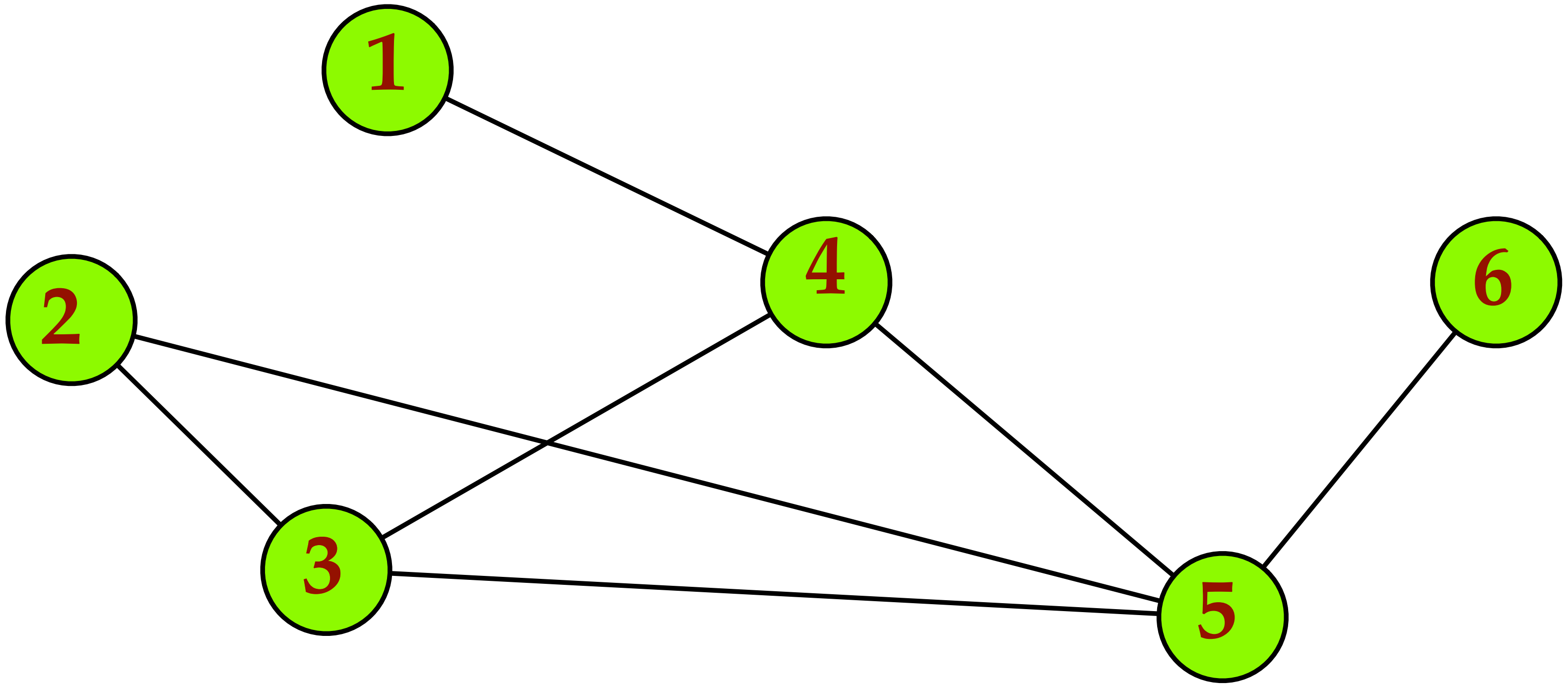
Def. An **automorphism** is an isomorphism from a graph to itself.

Orbit of a node u is the set of nodes that u is mapped to under some automorphism

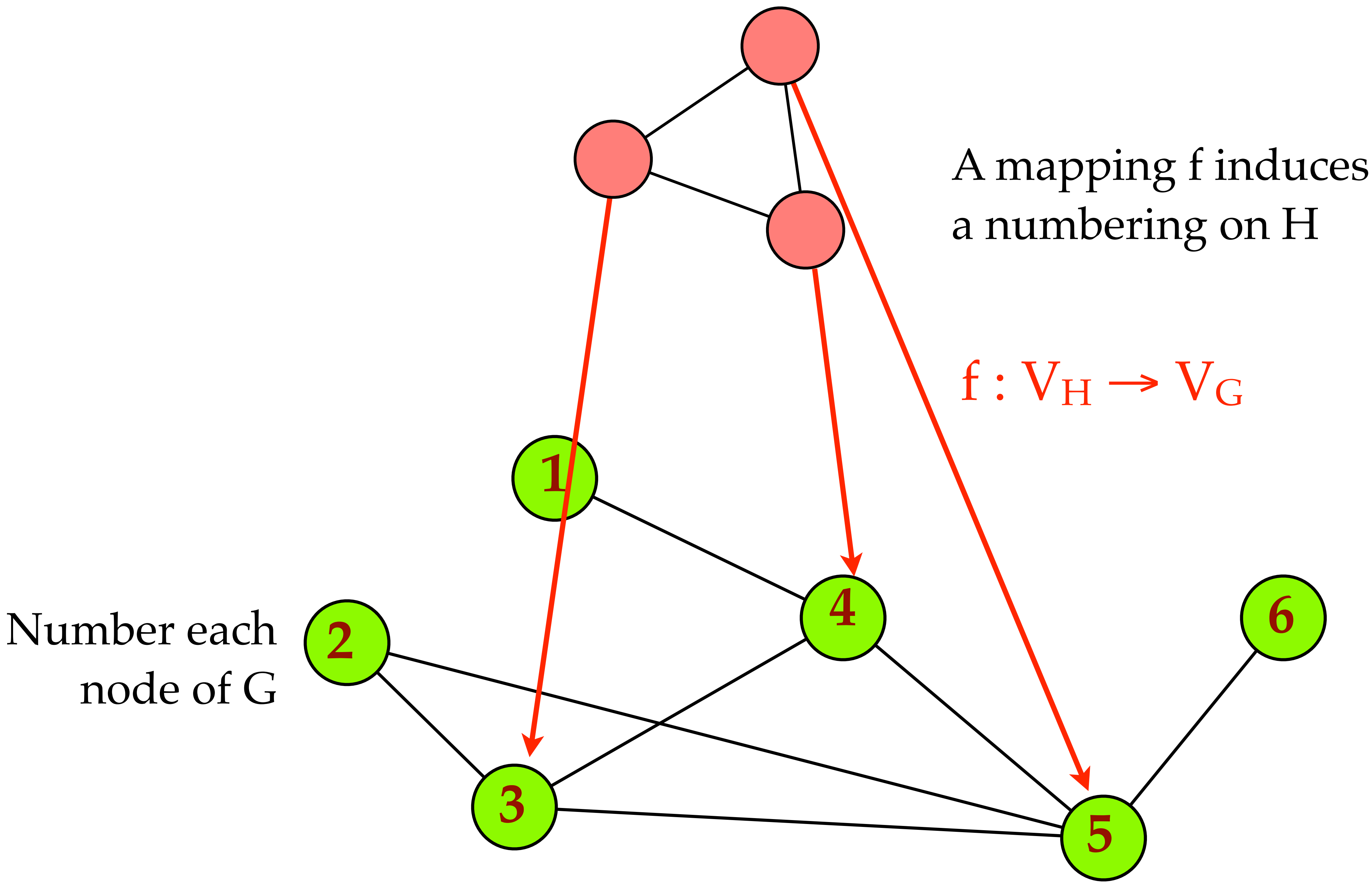


Main Speedup (#2)

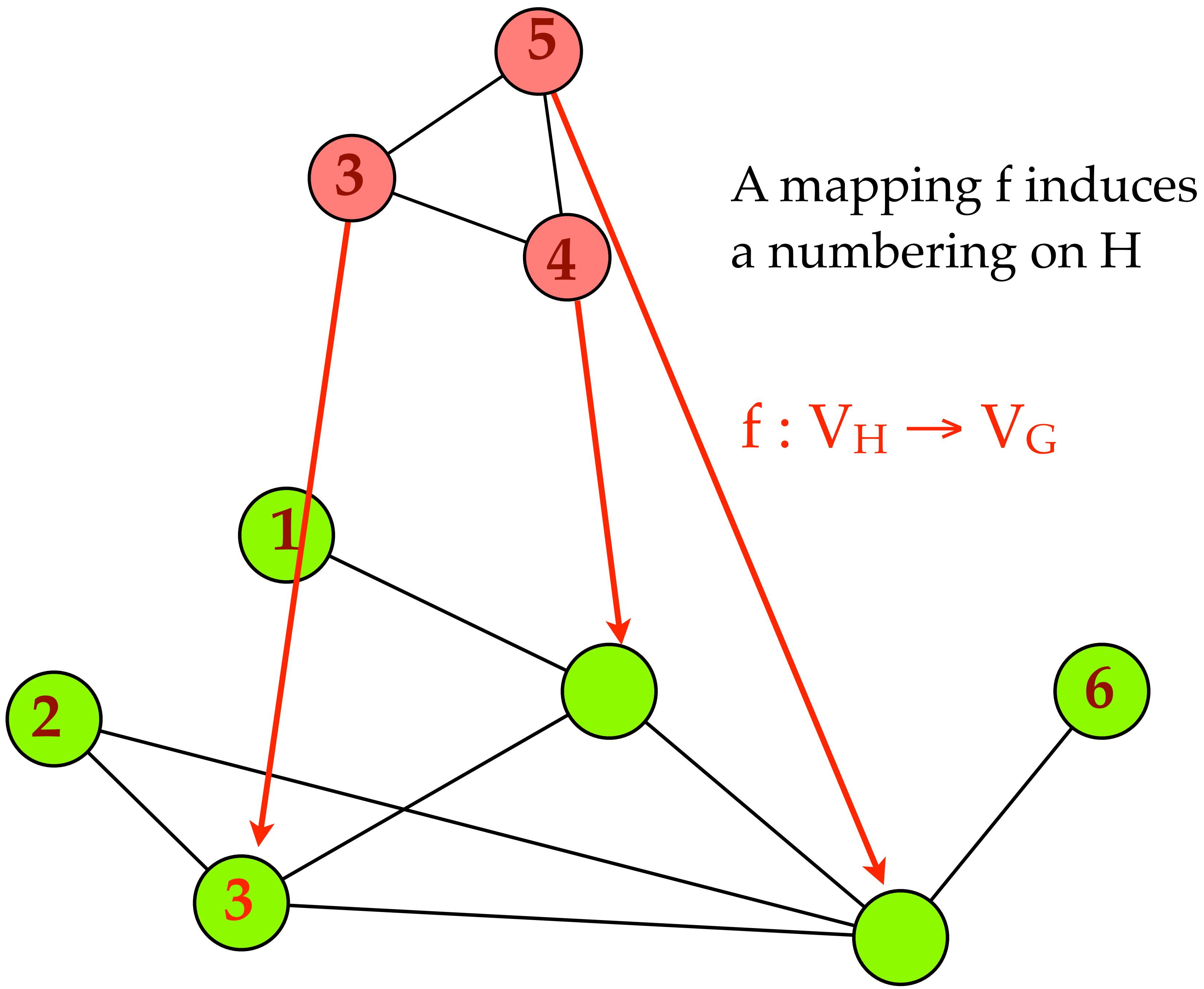
Number each
node of G



Main Speedup (#2)



Main Speedup (#2)



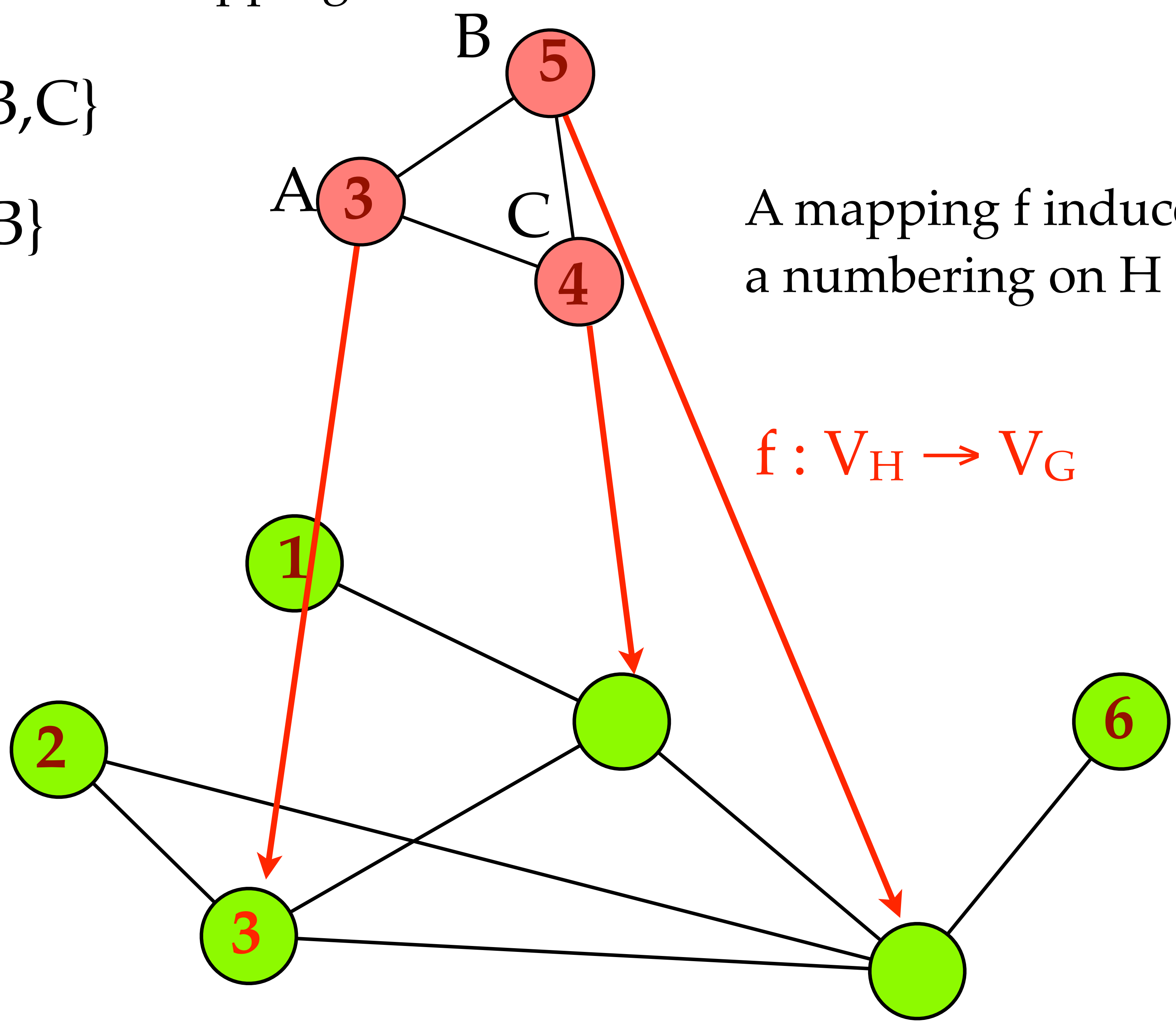
Number each
node of G

Main Speedup (#2)

If we add these constraints, we get only one possible mapping

A < min{B,C}

C < min{B}

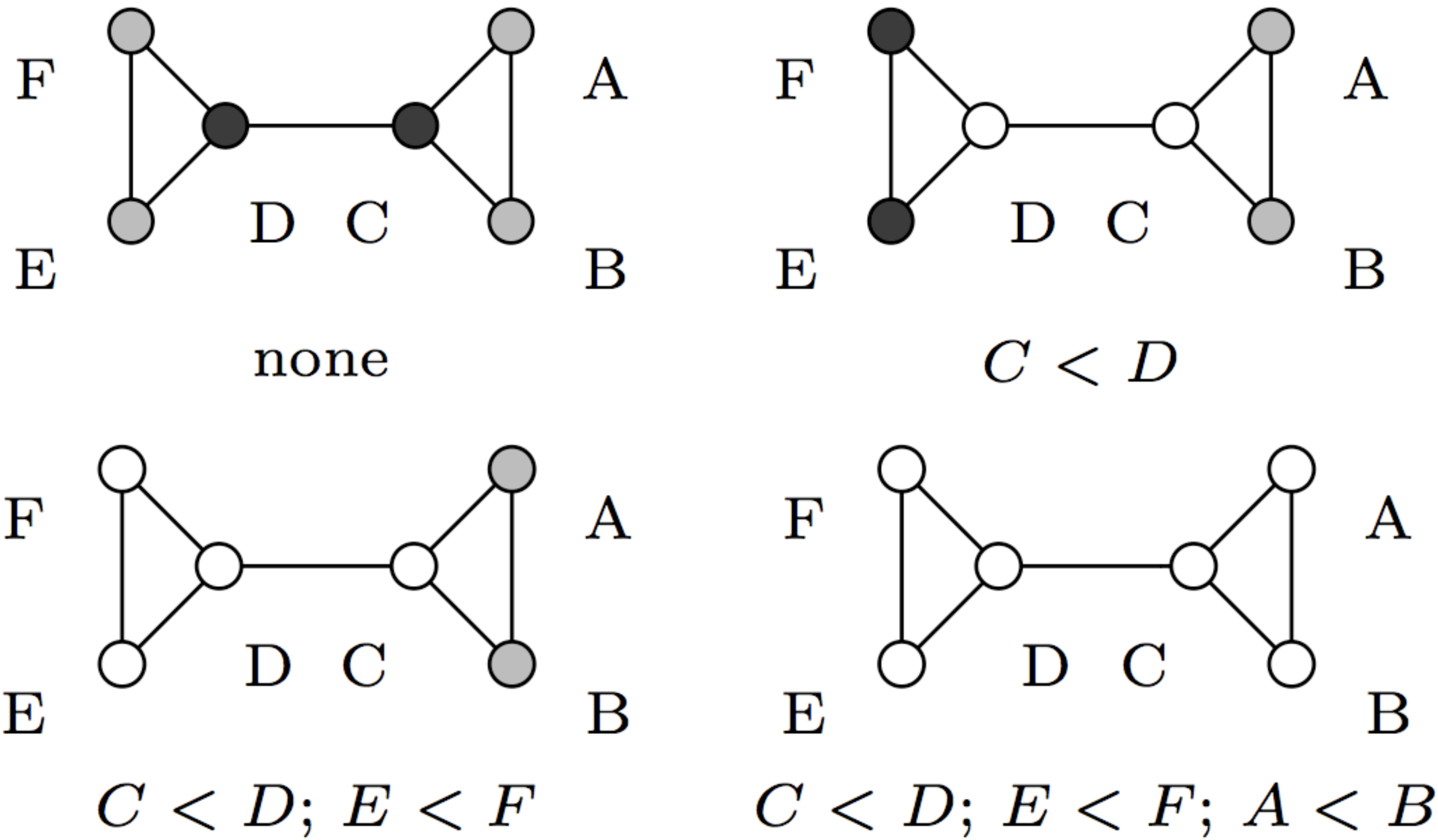


A mapping f induces a numbering on H

$f : V_H \rightarrow V_G$

Number each node of G

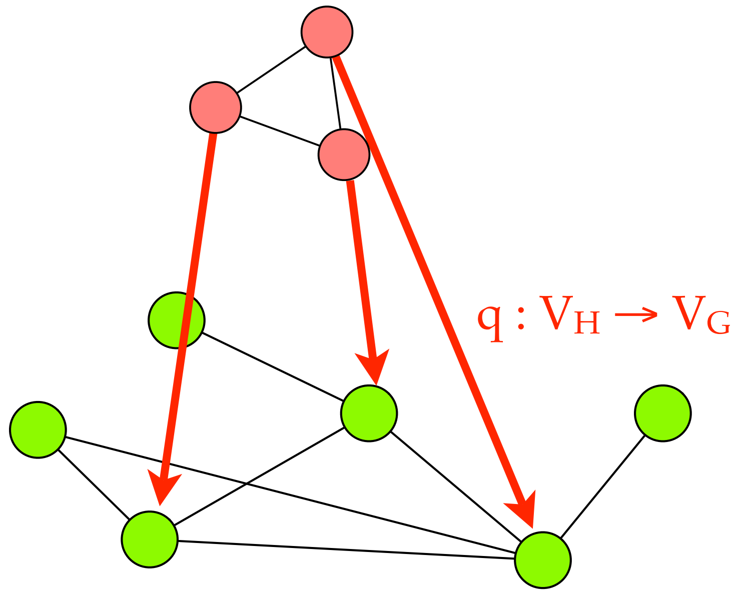
Adding Constraints, Larger Example



(Figure from Grochow & Kellis, 2007)

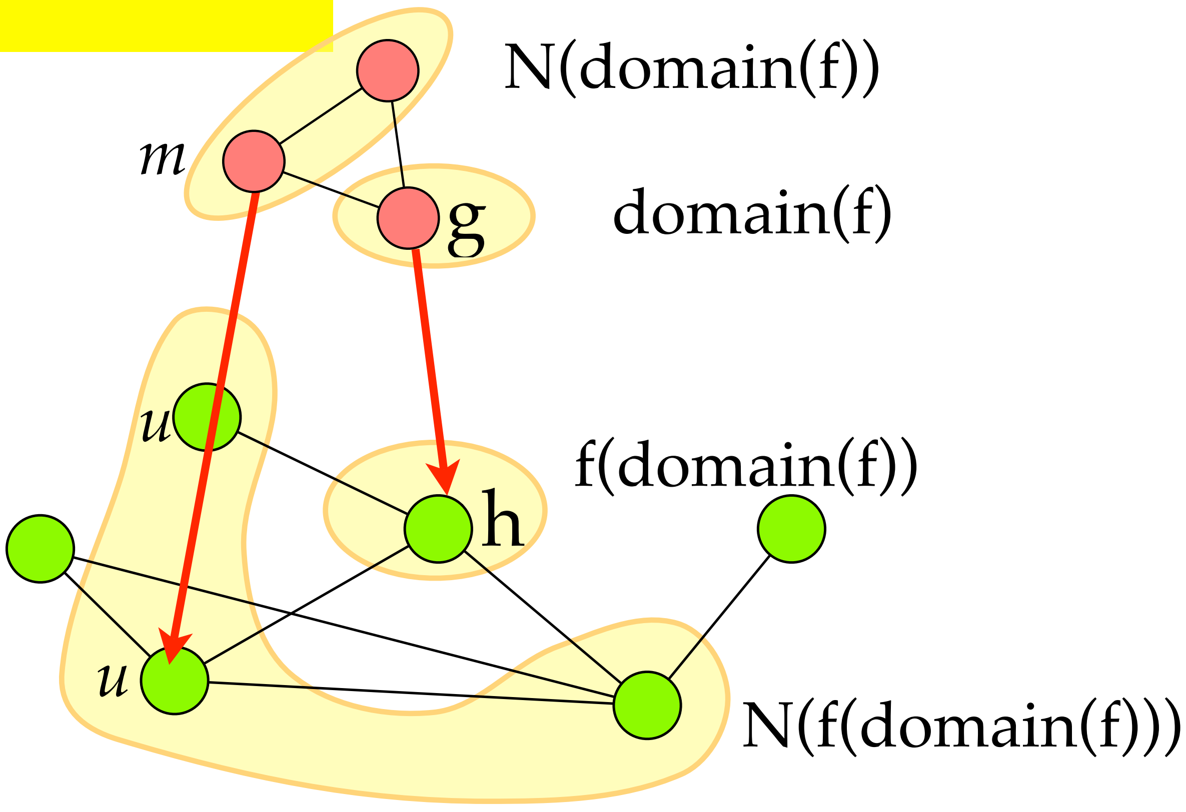
Basic Algorithm, differences for symmetry breaking

```
For each node g ∈ G
  For each node h ∈ H s.t. we haven't
  considered q ∈ Orbit(h):
    If h can't support g: continue

    Let f = {(g→h)}
    L = Extend(f, G, H, C_H)
    For q in L:
      Output image of q
Remove g from G
```

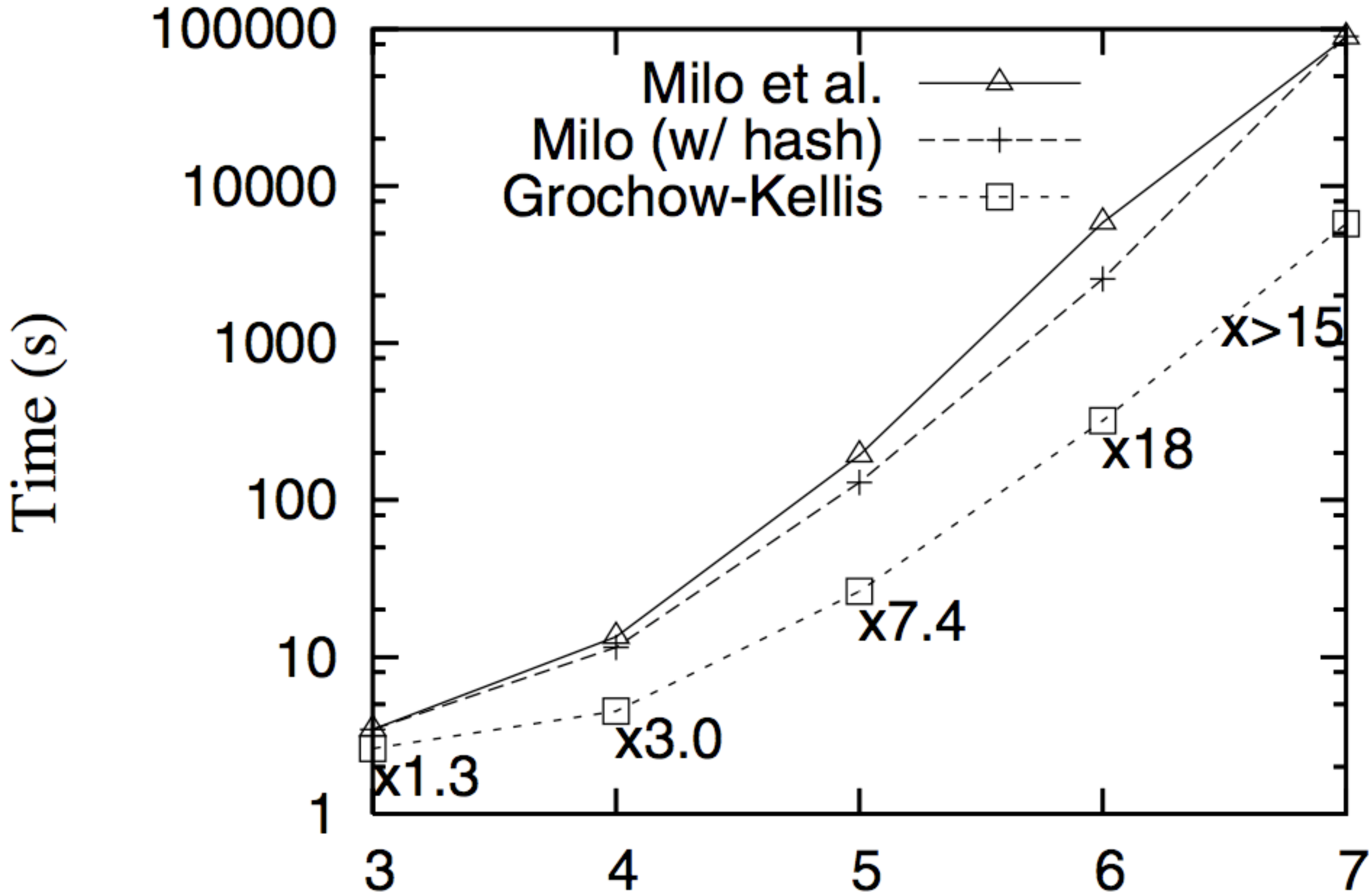


Extend(f, G, H), symmetry breaking differences

```
If domain(f) = H: return [f]

Let m = some node in N(domain(f))
For each node u ∈ N(f(domain(f))):

  If adding (m→u) to f keeps f as a
  valid isomorphism
  and (m→u) obeys the constraints
  then:
    Extend(f∪{(m→u)}, G, H)
```



Results: Running Time



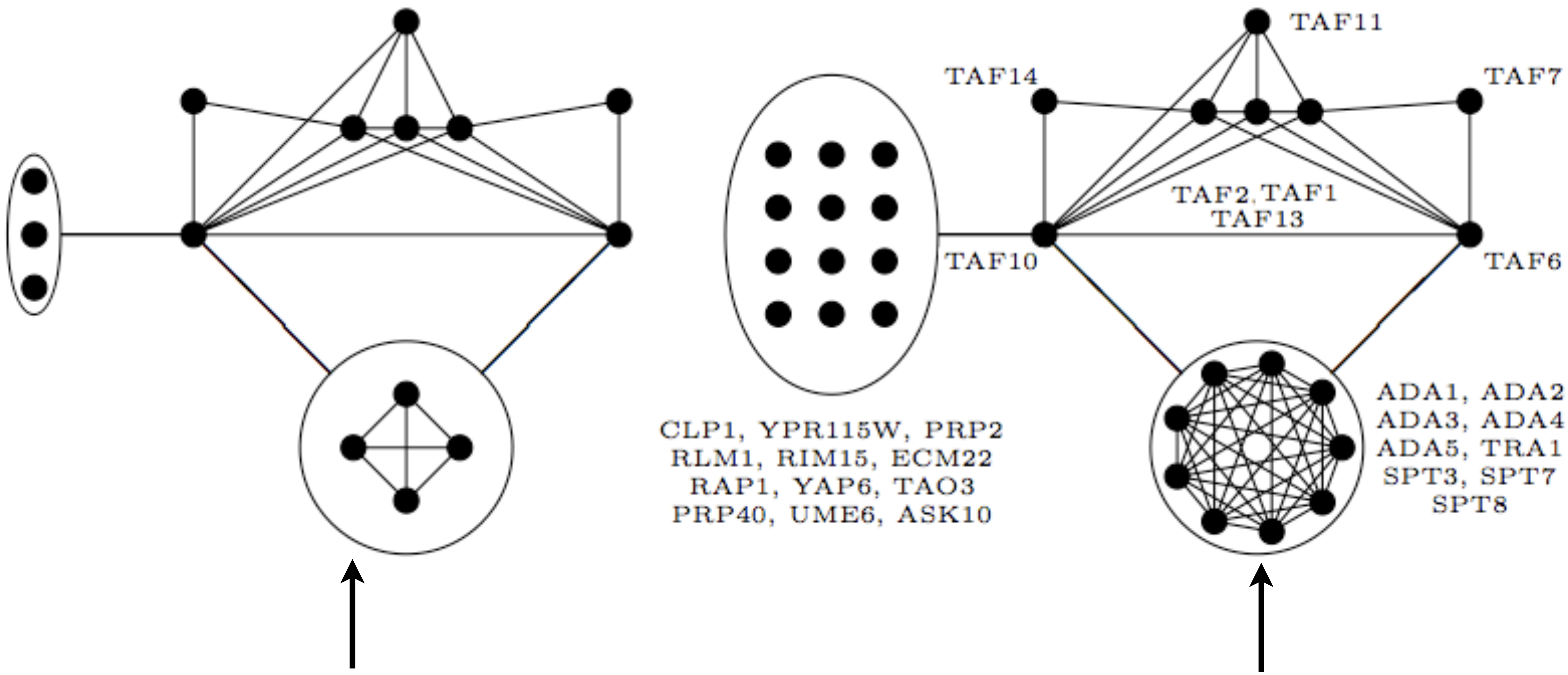
(Figure from Grochow & Kellis, 2007)

Results: Benefit of Symmetry Breaking

Nodes	Undirected PPI Network			Directed Regulatory Network		
	Total Subgraphs Searched	With Symmetry-Breaking	Improvement	Total Subgraphs Searched	With Symmetry-Breaking	Improvement
3	3.7×10^4	1.1×10^4	$\times3.13$	2.6×10^4	1.3×10^4	$\times2.02$
4	4.0×10^5	7.0×10^4	$\times5.77$	9.7×10^5	1.8×10^5	$\times5.41$
5	4.4×10^6	4.1×10^5	$\times10.9$	4.4×10^7	2.5×10^6	$\times18.0$
6	5.1×10^7	2.3×10^6	$\times22.2$	2.3×10^9	3.2×10^7	$\times73.3$
7	5.7×10^8	1.2×10^7	$\times46.3$	1.3×10^{11}	4.0×10^8	$\times334$
8	6.4×10^9	6.6×10^7	$\times96.2$	—	—	—

Really Large "motifs"? Meaningful?

(Figure from Grochow & Kellis, 2007)



Occurred 27,720 times in the real yeast PPI network (but rarely in a random network)

Really just a subgraph of this part of the yeast PPI: choose 4 nodes from the clique and 3 nodes from the oval.

Other Advantages

- Since symmetry breaking ensures each match is output only once, they don’t need to keep track of which graphs they’ve already output
  - save a lot of space
- Can be parallelized better

Spiritual Similarity to Color Coding

- Color Coding: make distinguishable things looks the same

- Symmetry Breaking: make indistinguishable things look different.