Parallel k-clique counting via Hierarchical Breadth First Search

Kalsuda Lapborisuth, Rajarshi Ghoshal, Srinivas Aluru

Georgia Institute of Technology





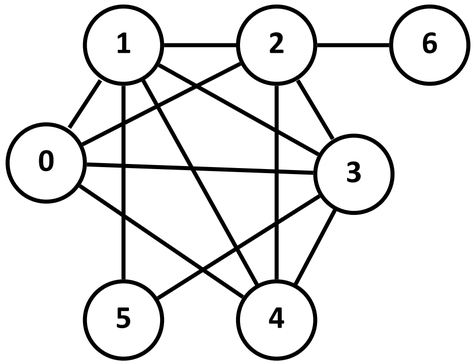


SC26 CHICAGO NOV 15–20

k-Clique Counting

Input: Graph $G(V, E)$ *and integer* $k \geq 3$,

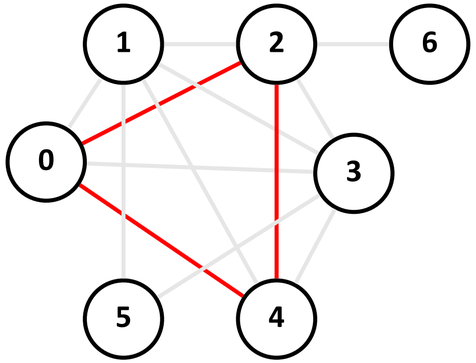
Output: The number of k-cliques



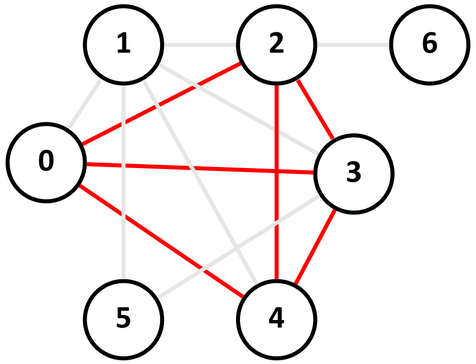
Types of Cliques

k-clique: A fully connected subgraph with $k \geq 3$ vertices

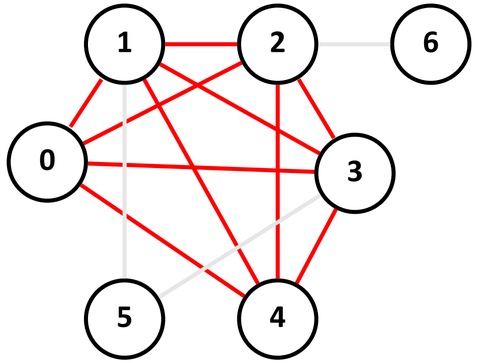
Maximal clique: A clique that cannot be increased further



3-cliques

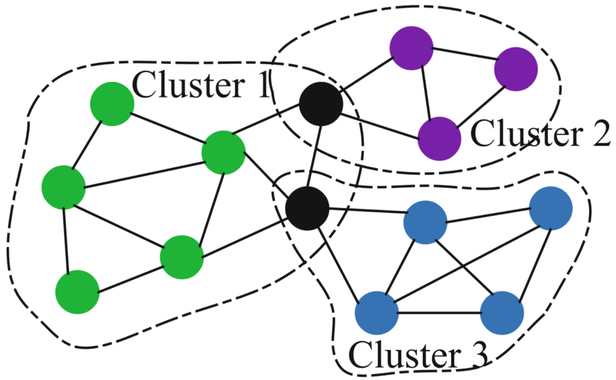


4-cliques



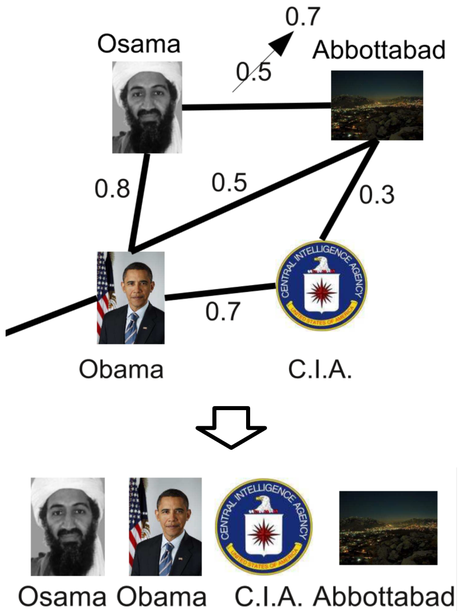
5-cliques

Applications

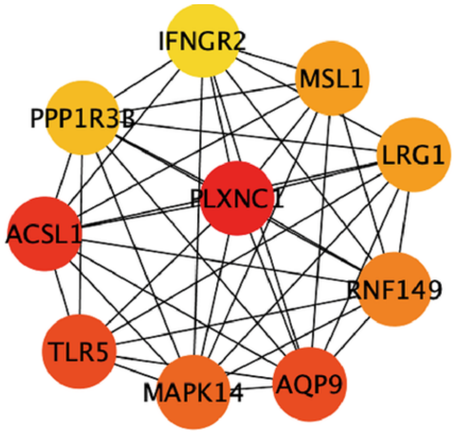


3-clique clusters

Community Detection[1]



Real-time Story Identification[2]



Correlated Genes Analysis[3]

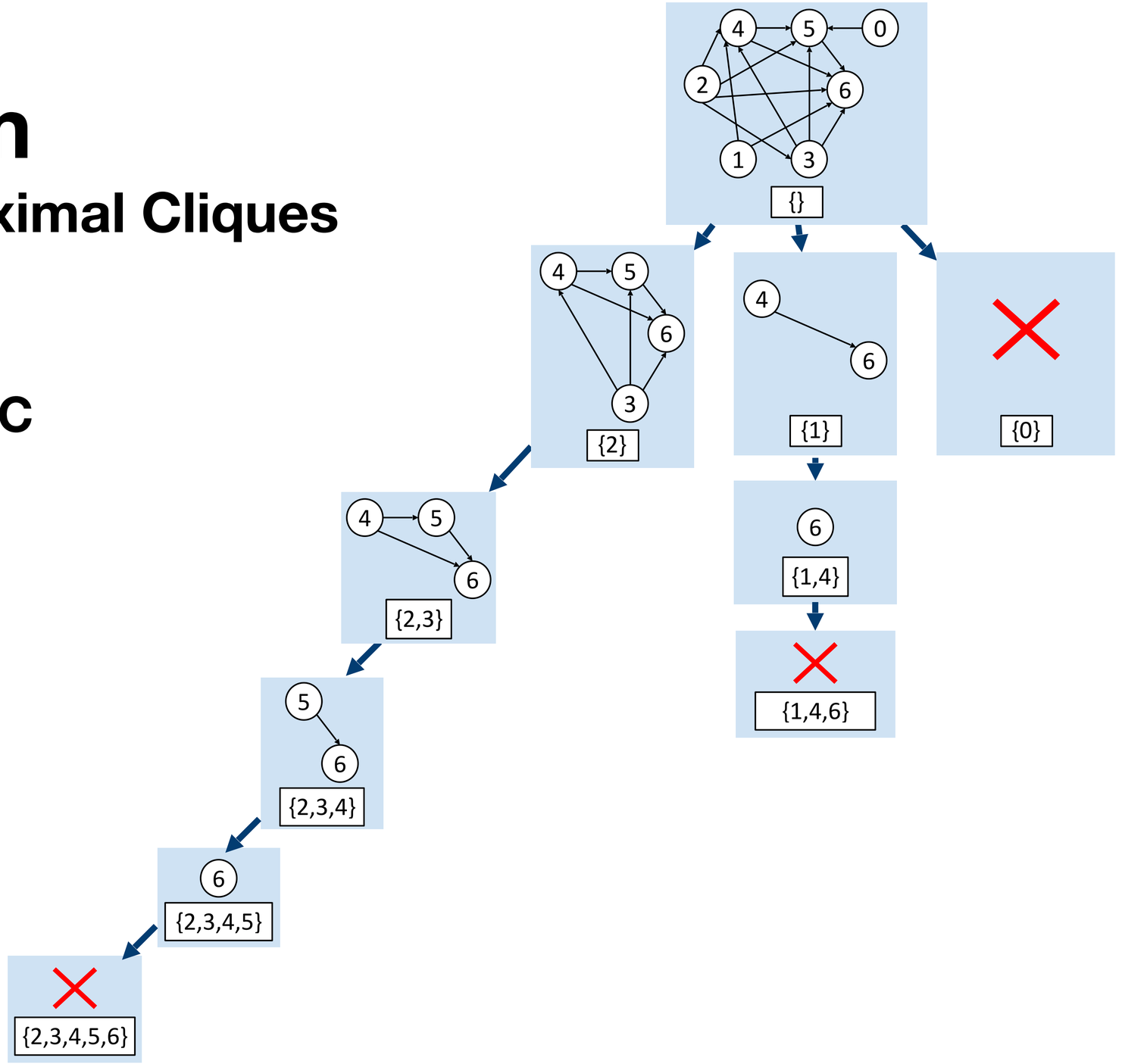
1. Hao et al., 2017. DOI: 10.1109/JSYST.2015.2433294
2. Angel et al., 2014. DOI: 10.1007/s00778-013-0340-z
3. Wang et al., 2023. DOI: 10.1049/syb2.12079

Pivoting Algorithm

Count k-cliques within Maximal Cliques

Each **branching** builds a clique **C** incrementally

1. Add a vertex to **C** from graph **G**
2. Removes vertices from **G** disconnected from **C**

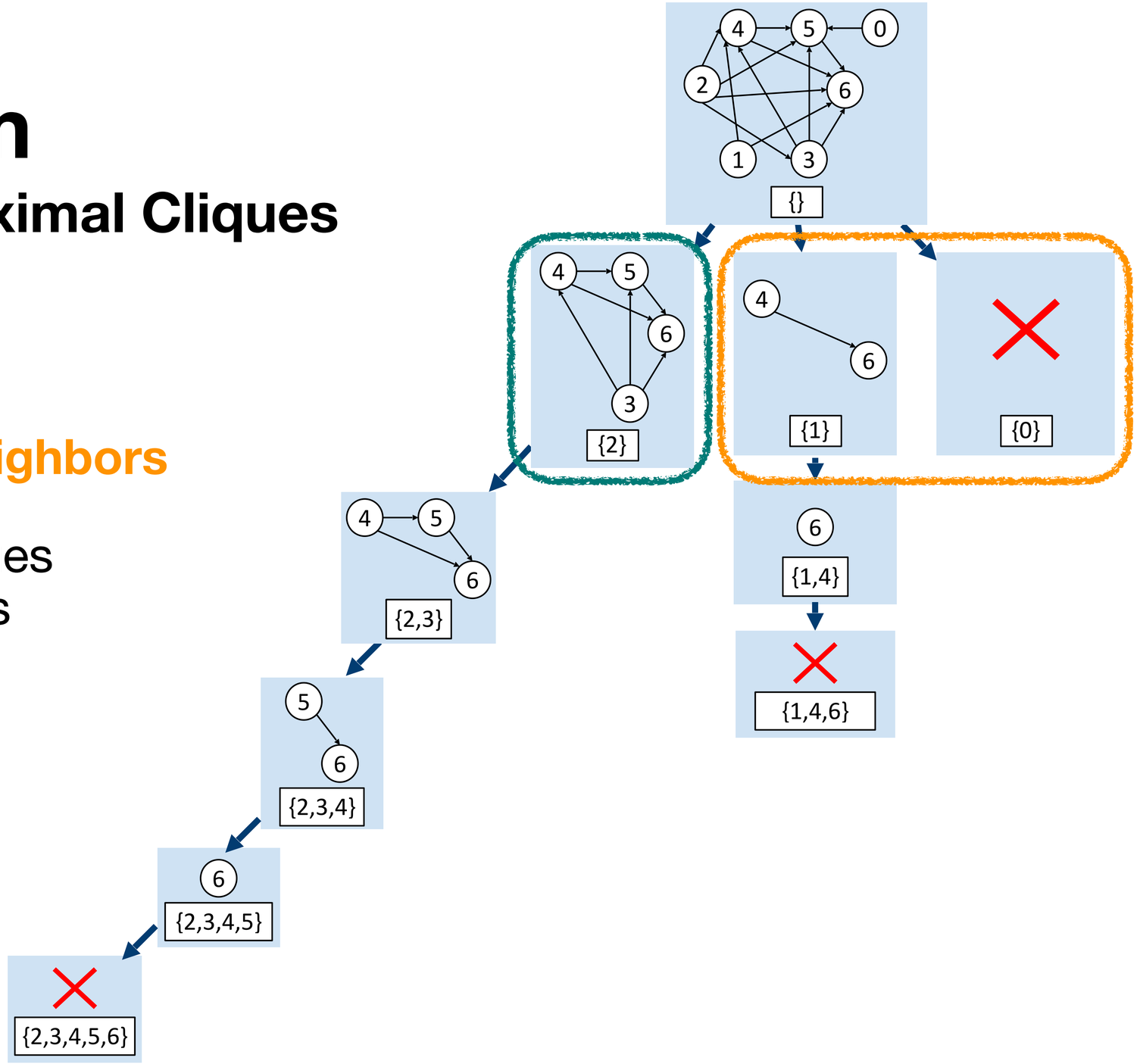


Pivoting Algorithm

Count k-cliques within Maximal Cliques

Branch on **pivot** and its **non-neighbors**

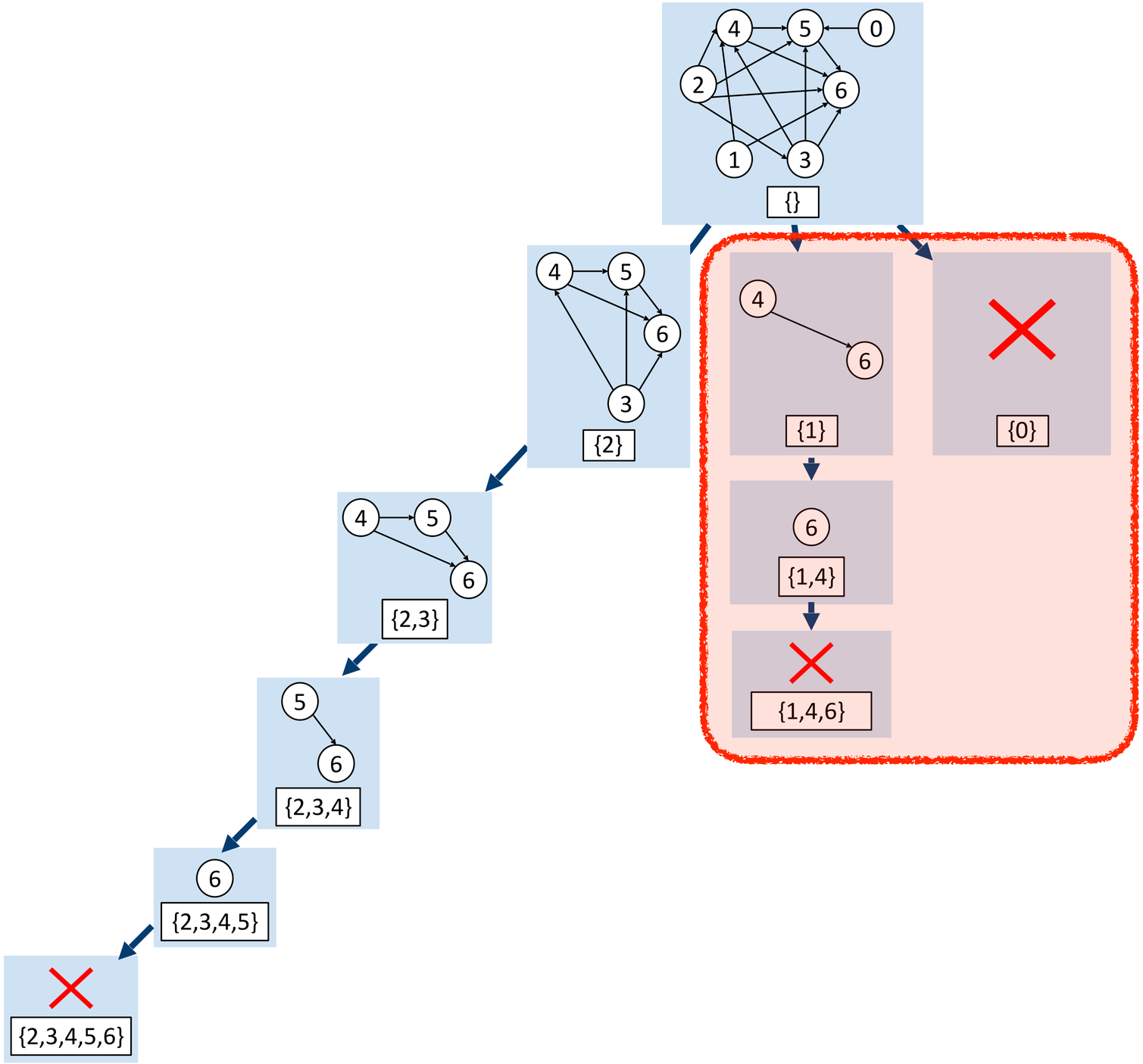
- The pivot branch finds cliques containing pivot's neighbors
- Max degree vertex for pivot



Fat Search Tree

Issue 1

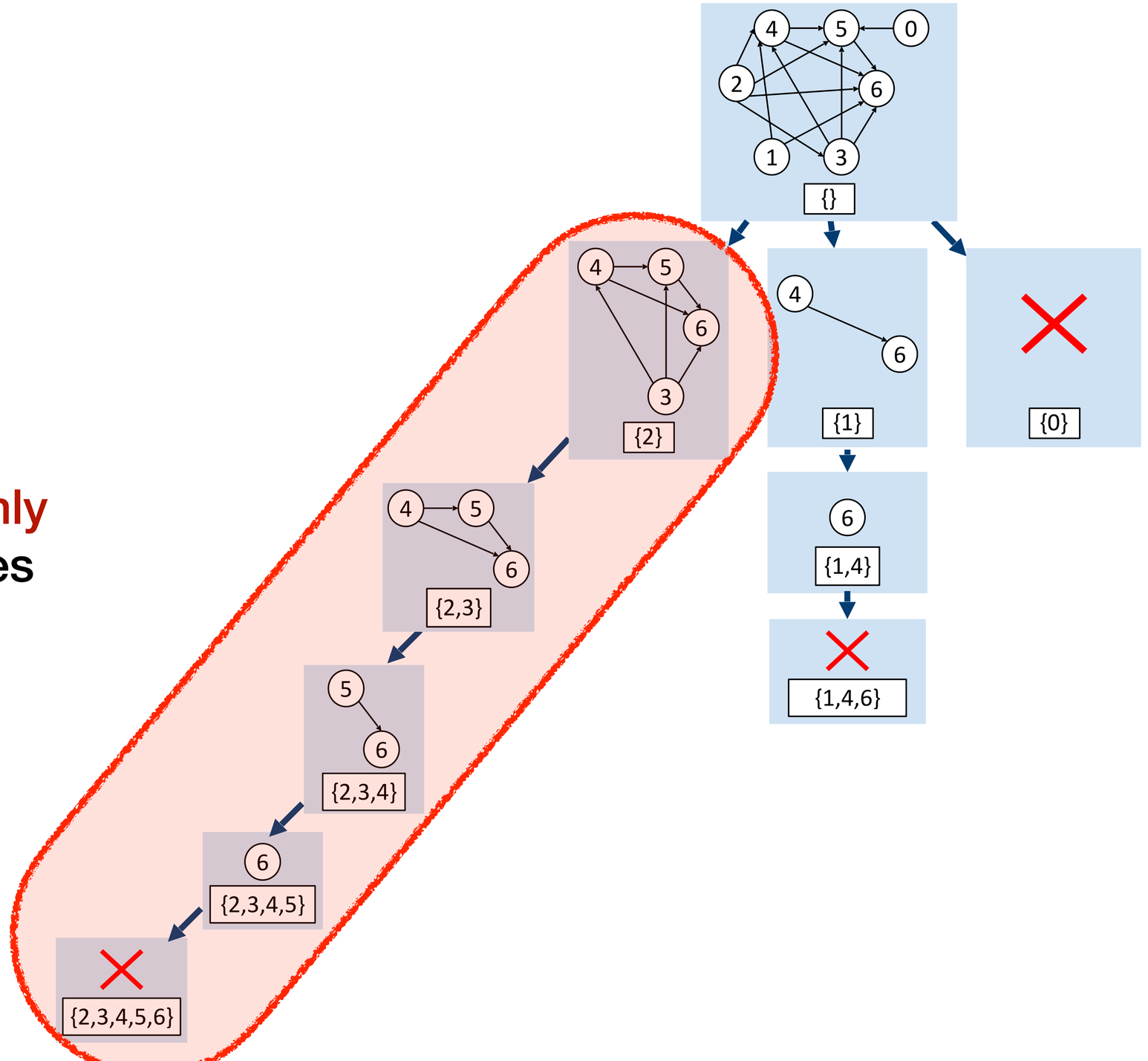
Wasted work exploring subproblems that contain no k-cliques



Tall Search Tree

Issue 2

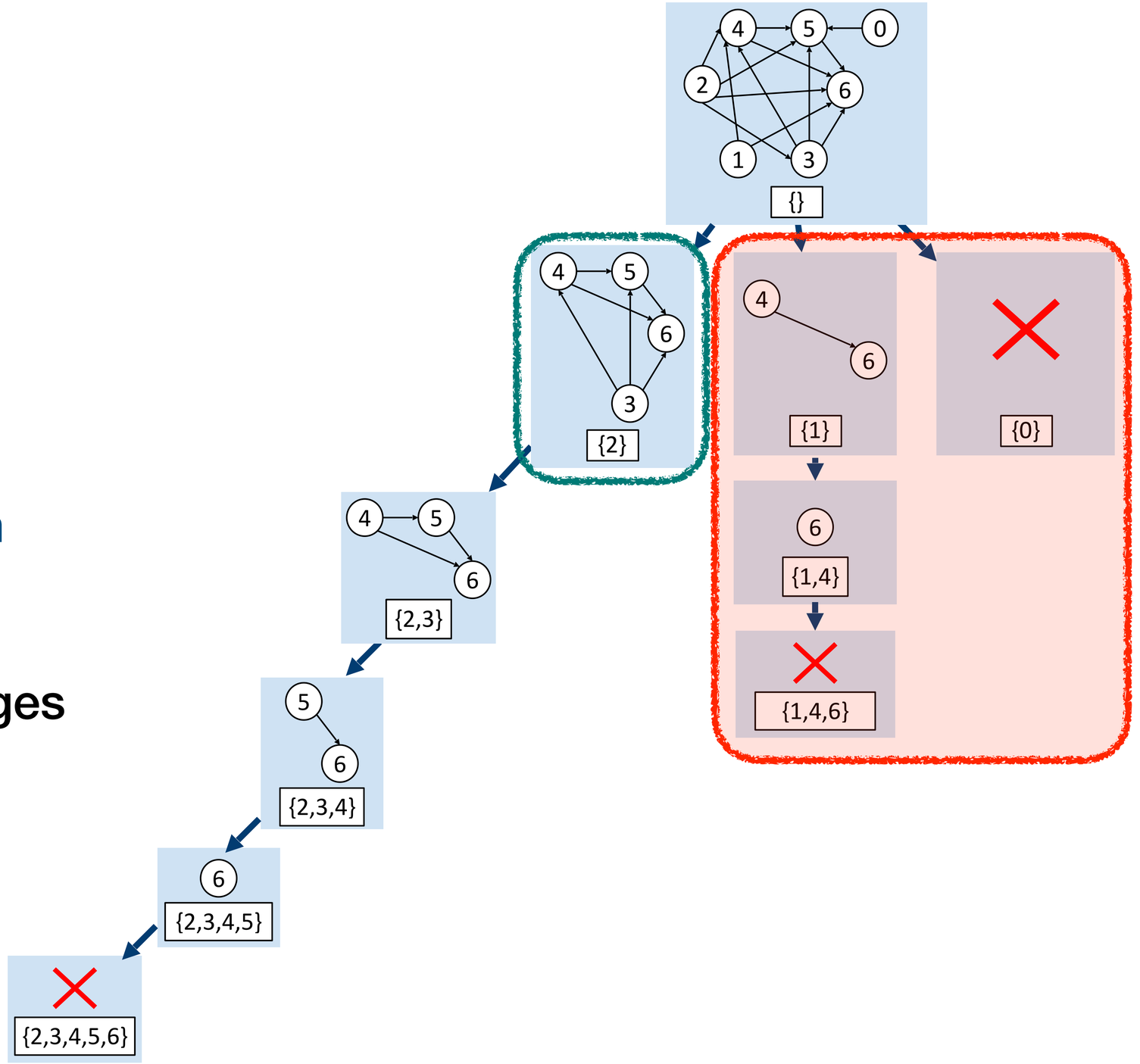
Load imbalance from highly irregular subproblem sizes



HitSearch

Issue 1: Fat Search Tree

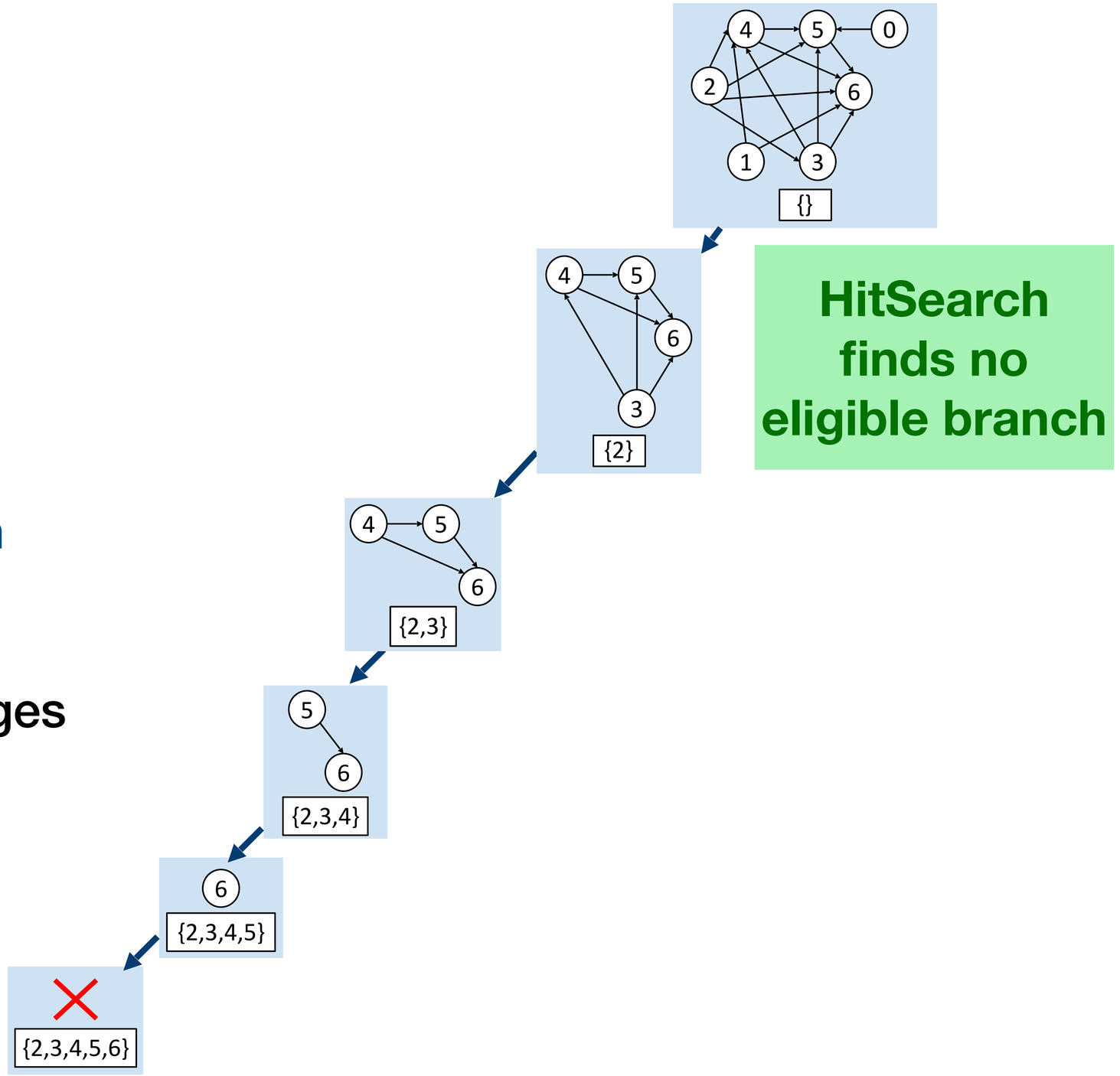
Hierarchical BFS Algorithm
to restrict branching
on non-neighbors
a **compact** ***"hitting set"*** of edges



HitSearch

Issue 1: Fat Search Tree

Hierarchical BFS Algorithm
to restrict branching
on non-neighbors
a **compact *"hitting set"*** of edges



HitSearch finds no eligible branch

Hitting Set

Thinner Search Tree

Hitting set: A set of edges intersects every k-clique

Smaller Hitting Set

↓

Less Branching

↓

Faster Runtime

Hitting Set

Extending the Pivoting Algorithm

Function HitSearch(G, k, C, P)**:**

- **if** $|V(G)| = 0$ ***or*** $|C| - |P| = k$ **then** **return** $\binom{|P|}{k-|C|-|P|}$;
- $count \leftarrow 0$
- $p \leftarrow$ FindPivot(G) [15]
- $G_p \leftarrow$ NeighborSubgraph(G, p)
- $count$ += HitSearch(G_p, k, $C \cup \{p\}$, $P \cup \{p\}$)
- **for** $w \in \{V(G) \setminus (N(p) \cup \{p\})\}$ **do**
  - $G_w \leftarrow$ NeighborSubgraph(G, w)
  - $count$ += HitSearch(G_w, k, $C \cup \{w\}$, P)
- **return** $count$

Hitting Set

Extending the Pivoting Algorithm

Function HitSearch(G, k, C, P):
  if $|V(G)| = 0$ ***or*** $|C| - |P| = k$ **then** **return** $\binom{|P|}{k-|C|-|P|}$;
  $count \leftarrow 0$
  $p \leftarrow$ FindPivot(G) [15]
  $G_p \leftarrow$ NeighborSubgraph(G, p)
  $count$ += HitSearch(G_p, k, $C \cup \{p\}$, $P \cup \{p\}$)
  if $(k - (|C| - |P|) >= 3) \wedge (k - |C| >= 3)$ **then**
    $\mathcal{H} \leftarrow$ FindHittingSet(G, p, k–$|C|$) // III-B
    for $e = (u, v) \in \mathcal{H}$ **do**
      $G_e \leftarrow$ HitEdgeSubgraph(G, e, $\mathcal{H}$) // III-C
      $count$ += HitSearch(G_e, k, $C \cup \{u, v\}$, P)
  else
    for $w \in \{V(G) \setminus (N(p) \cup \{p\})\}$ **do**
      $G_w \leftarrow$ NeighborSubgraph(G, w)
      $count$ += HitSearch(G_w, k, $C \cup \{w\}$, P)
  return $count$

Hitting Set

And how to find it?

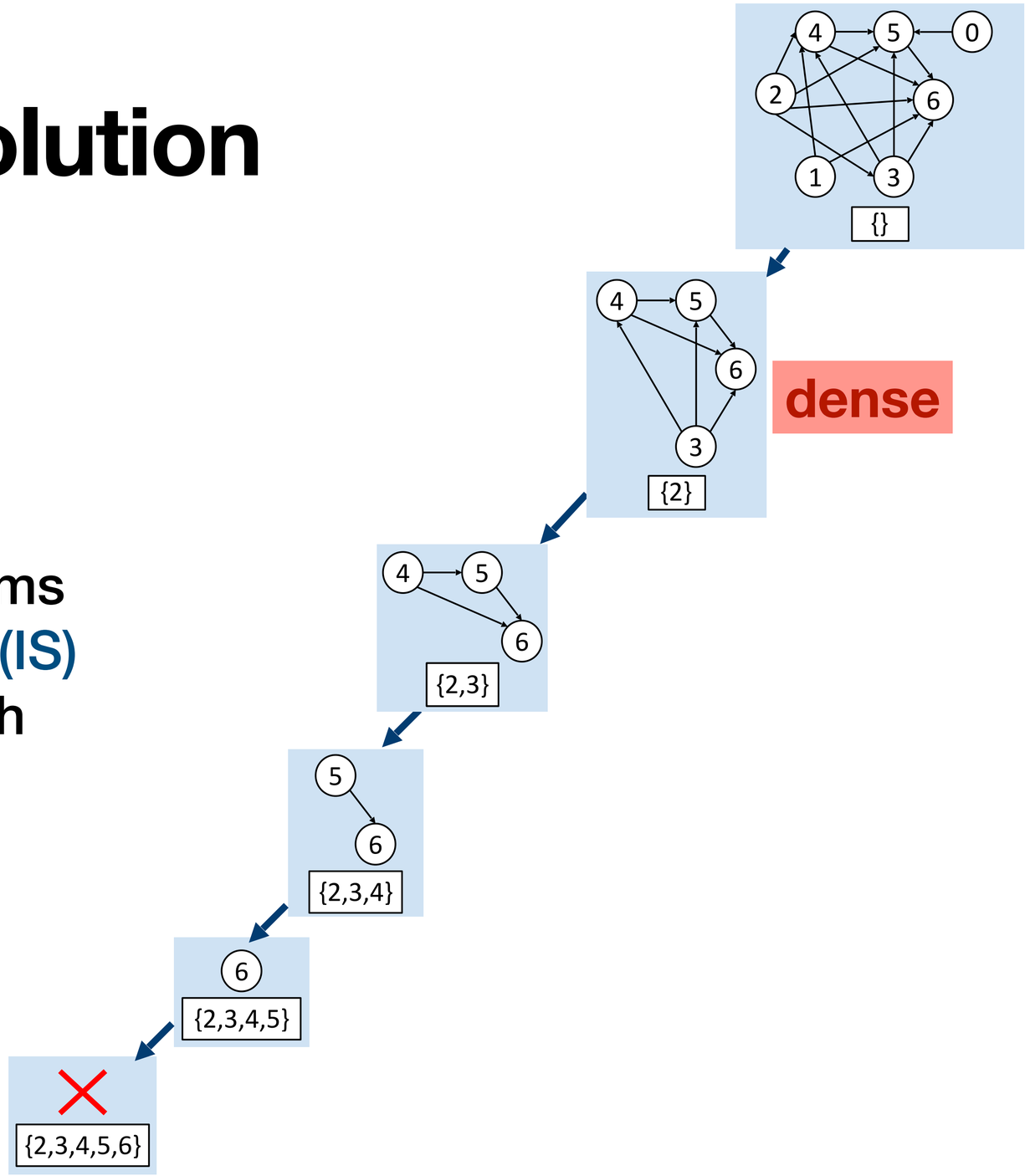
Hierarchical BFS Algorithm
for **low cost** construction
at every branching

Function `FindHittingSet`(G, p, k):
  $\ell_1 \leftarrow$ `BFS`(G, p); `// use pivot as source`
  $k \leftarrow \lceil k/2 \rceil$; $d \leftarrow 1$
  while $k > 2$ **do**
    $\ell_{d+1} \leftarrow$ `BFSOnLevelEdges`(G, ℓ_d)
    $k \leftarrow \lceil k/2 \rceil$; $d \leftarrow d+1$
  $H_1 \leftarrow \{(u,v) \in E(G) \mid \{\ell_1(u), \ell_1(v)\} = \{1,2\}\}$
  $H_2 \leftarrow \{(u,v) \in E(G) \mid \ell_d(u) = \ell_d(v) \wedge \ell_1(u) \geq 2 \wedge \ell_1(v) \geq 2\}$;
  return $H_1 \cup H_2$

Calico - Hybrid Solution

Issue 2: Tall Search Tree

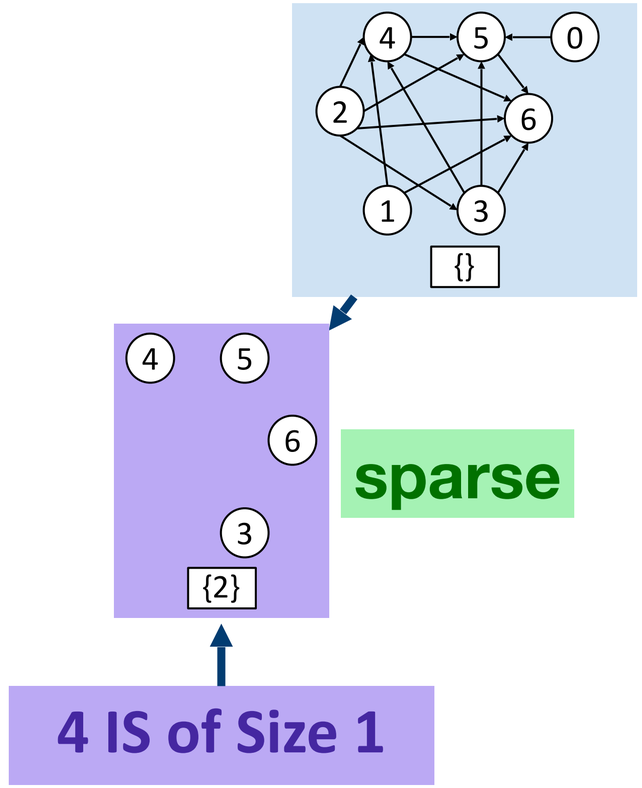
Reformulate **dense** subproblems
as counting Independent Sets (IS)
on **sparse** complement graph



Calico - Hybrid Solution

Issue 2: Tall Search Tree

Reformulate **dense** subproblems
as counting Independent Sets (IS)
on **sparse** complement graph



IS on Complement Graph

How to recover k-clique count with pivots?

Function `CountCliquesInDense`$(G,\ k,\ C,\ np)$**:**

$\bar{G} \leftarrow$ `BuildComplement`(G)

$IS \leftarrow$ `CountIS`$(\bar{G},\ k - (|C| - np))$

return $\sum_{j=0}^{np} \binom{np}{j} \cdot IS[k' - j]$

IS on Complement Graph

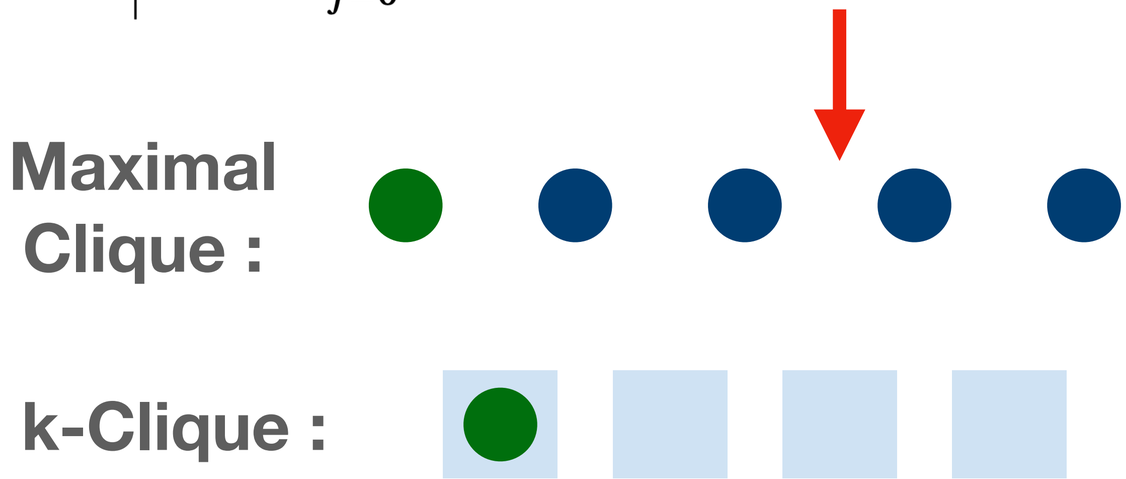
How to recover k-clique count with pivots?

Function CountCliquesInDense(G, k, C, np):

$\bar{G} \leftarrow$ BuildComplement(G)

$IS \leftarrow$ CountIS($\bar{G}$, $k - (|C| - np)$)

return $\sum_{j=0}^{np} \binom{np}{j} \cdot IS[k' - j]$



IS on Complement Graph

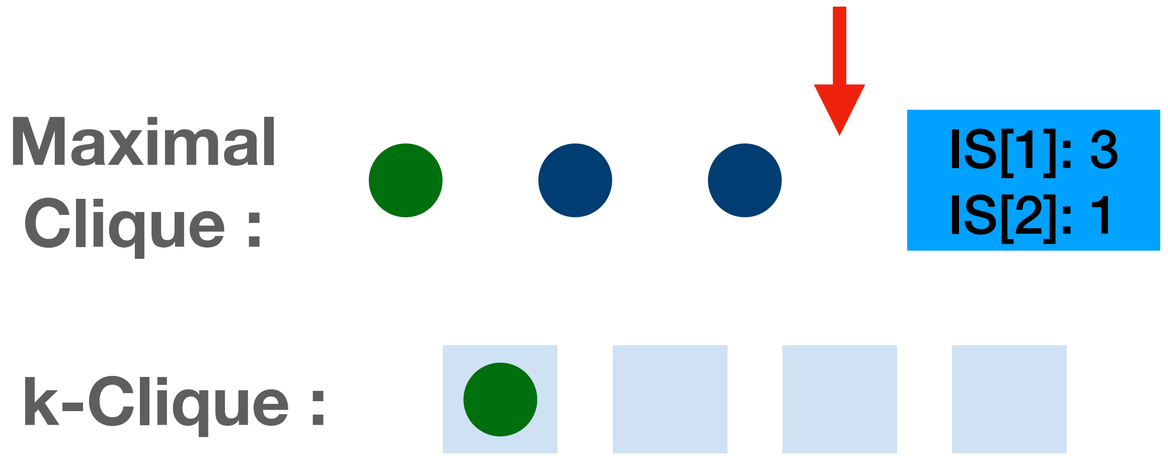
How to recover k-clique count with pivots?

Function CountCliquesInDense(G, k, C, np):

$\bar{G} \leftarrow$ BuildComplement(G)

$IS \leftarrow$ CountIS($\bar{G}$, $k - (|C| - np)$)

return $\sum_{j=0}^{np} \binom{np}{j} \cdot IS[k' - j]$



IS on Complement Graph

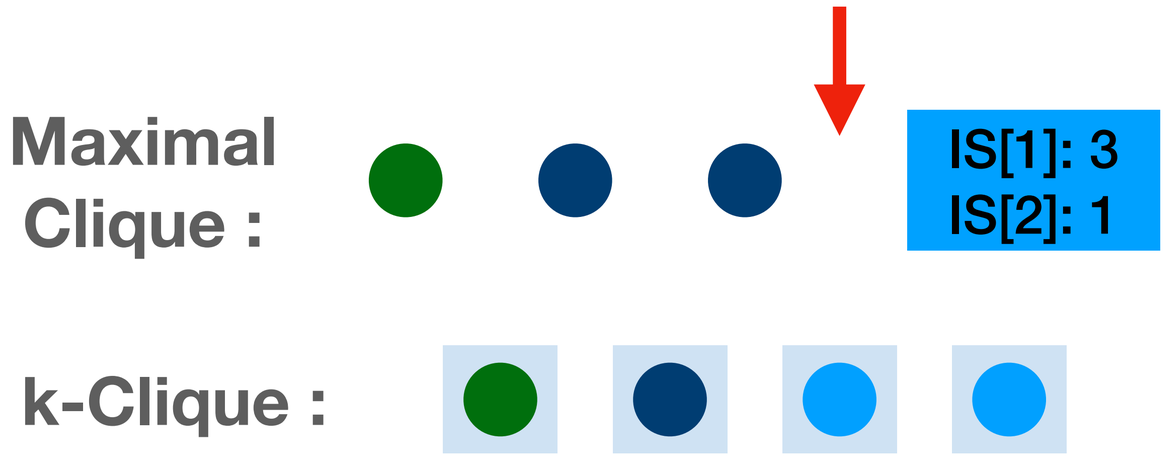
How to recover k-clique count with pivots?

Function CountCliquesInDense(G, k, C, np):

$\bar{G} \leftarrow$ BuildComplement(G)

$IS \leftarrow$ CountIS($\bar{G}$, $k - (|C| - np)$)

return $\sum_{j=0}^{np} \binom{np}{j} \cdot IS[k' - j]$



IS on Complement Graph

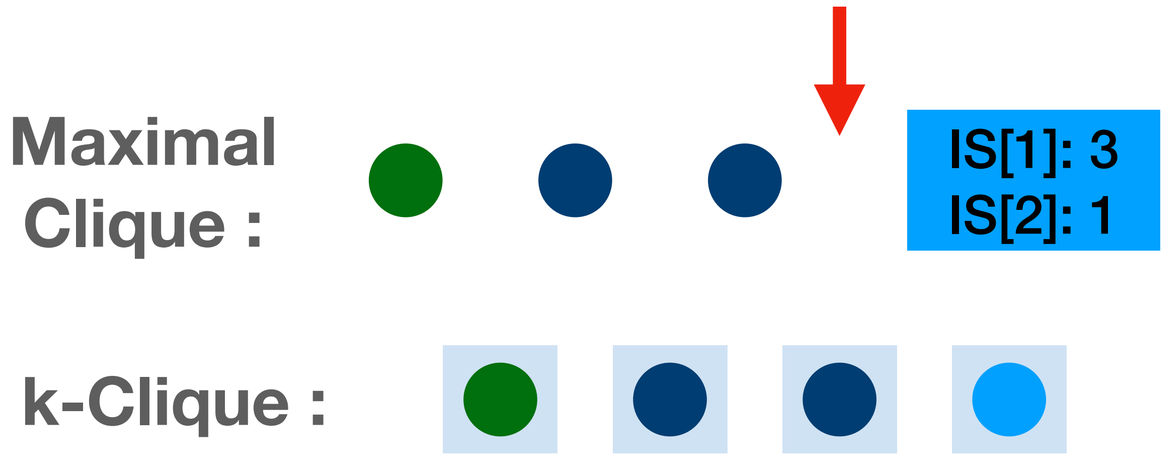
How to recover k-clique count with pivots?

Function CountCliquesInDense(G, k, C, np):

$\bar{G} \leftarrow$ BuildComplement(G)

$IS \leftarrow$ CountIS($\bar{G}$, $k - (|C| - np)$)

return $\sum_{j=0}^{np} \binom{np}{j} \cdot IS[k' - j]$

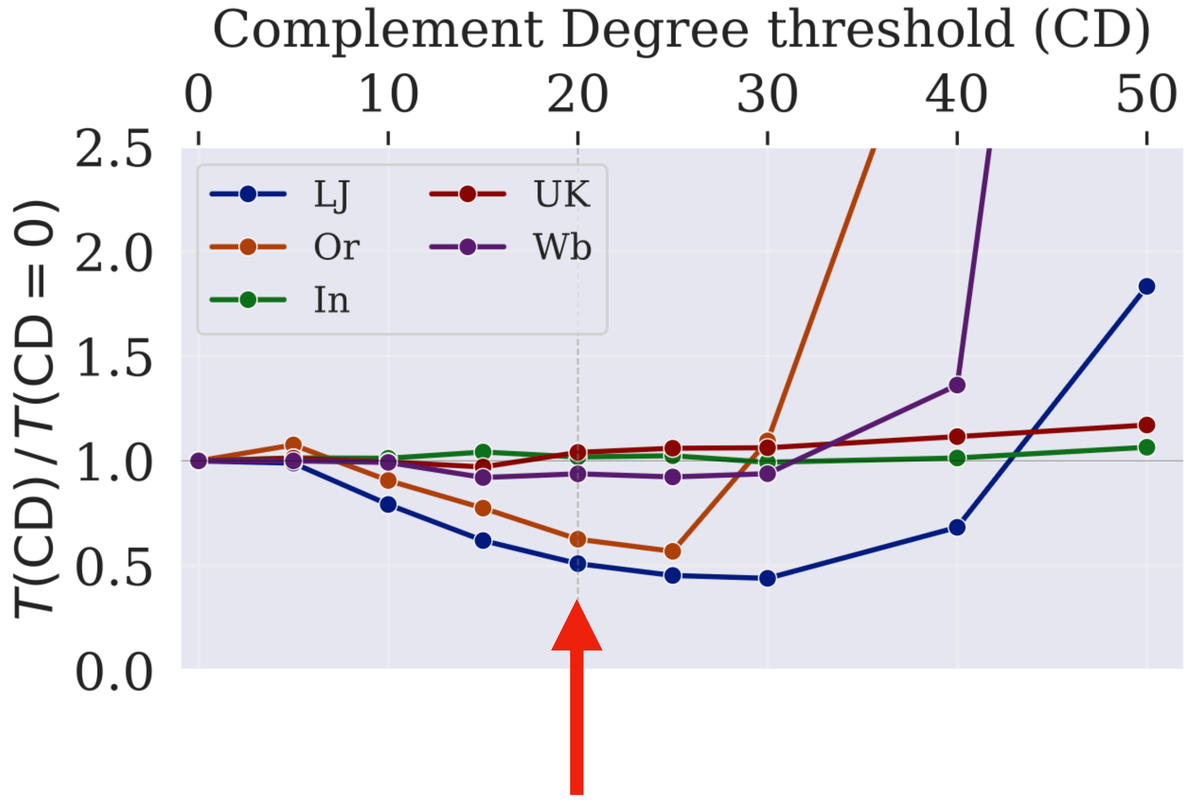


When to Switch to Counting IS?

Effects of Density Parameters (1)

Complement Graph $\bar{G}$ is sufficiently **sparse**

$$|E(\overline{G})| < CD\,|V(G)|$$

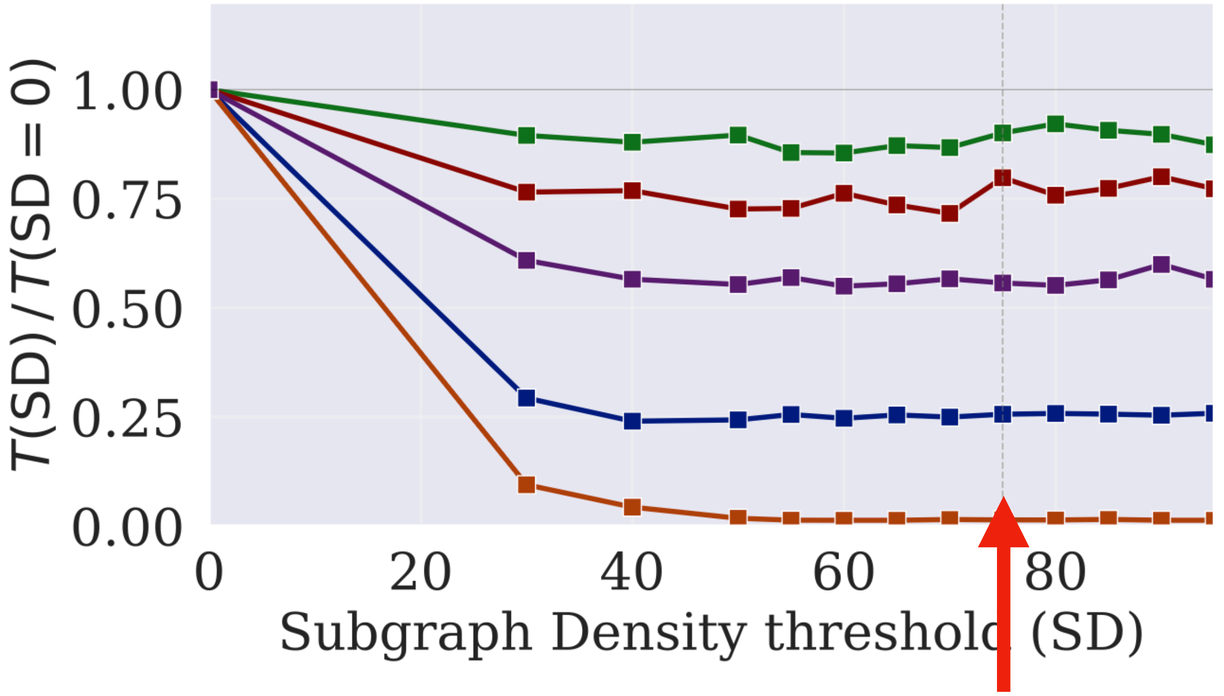


When to Switch to Counting IS ?

Effects of Density Parameters (1)

$$\frac{200|E(\bar{G})|}{|V(G)|(|V(G)|-1)} > SD$$

$\bar{G}$ is not as sparse but remaining **clique size is small**

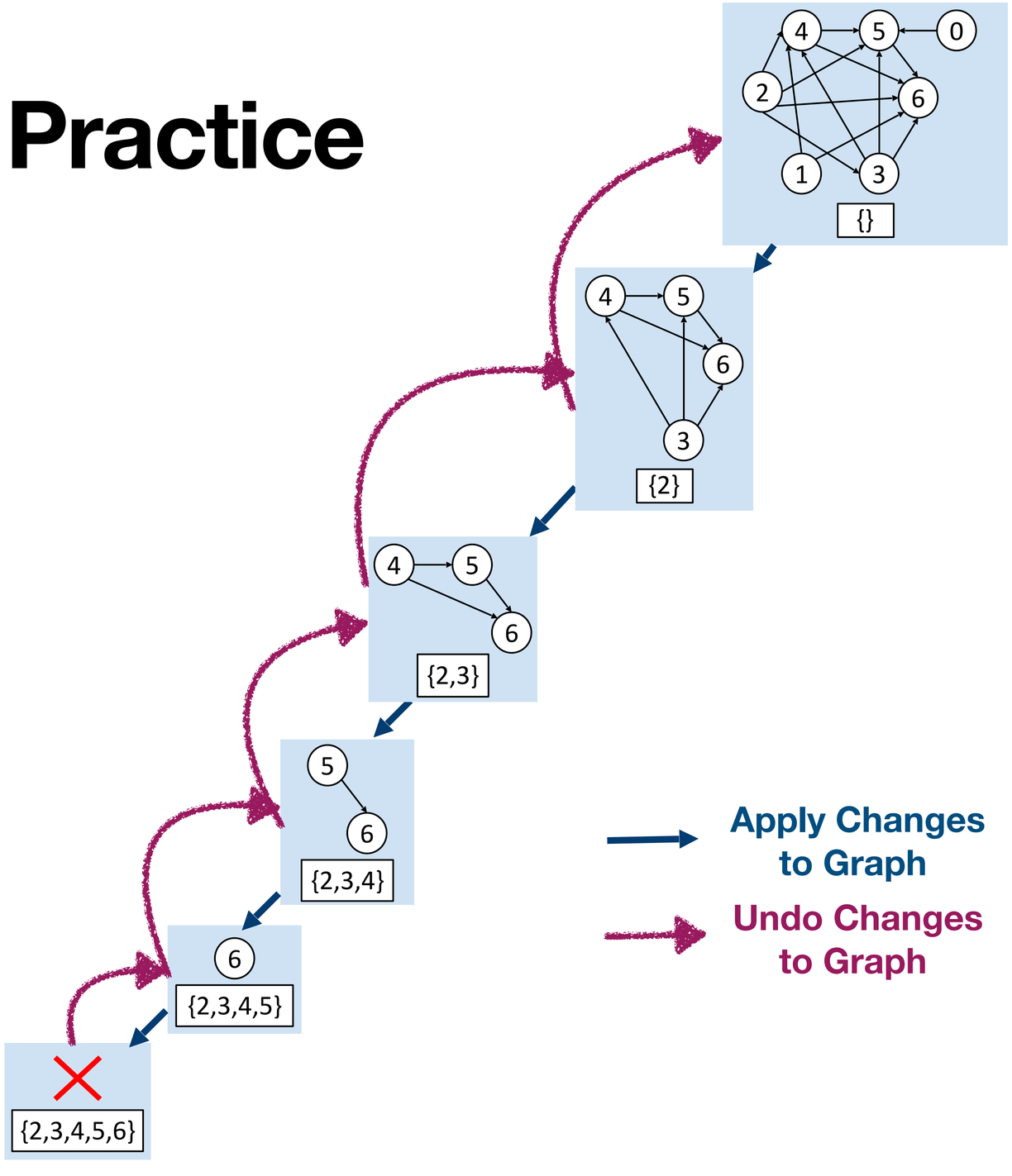


Scaling Challenges in Practice

Issue 1: In-place Backtracking

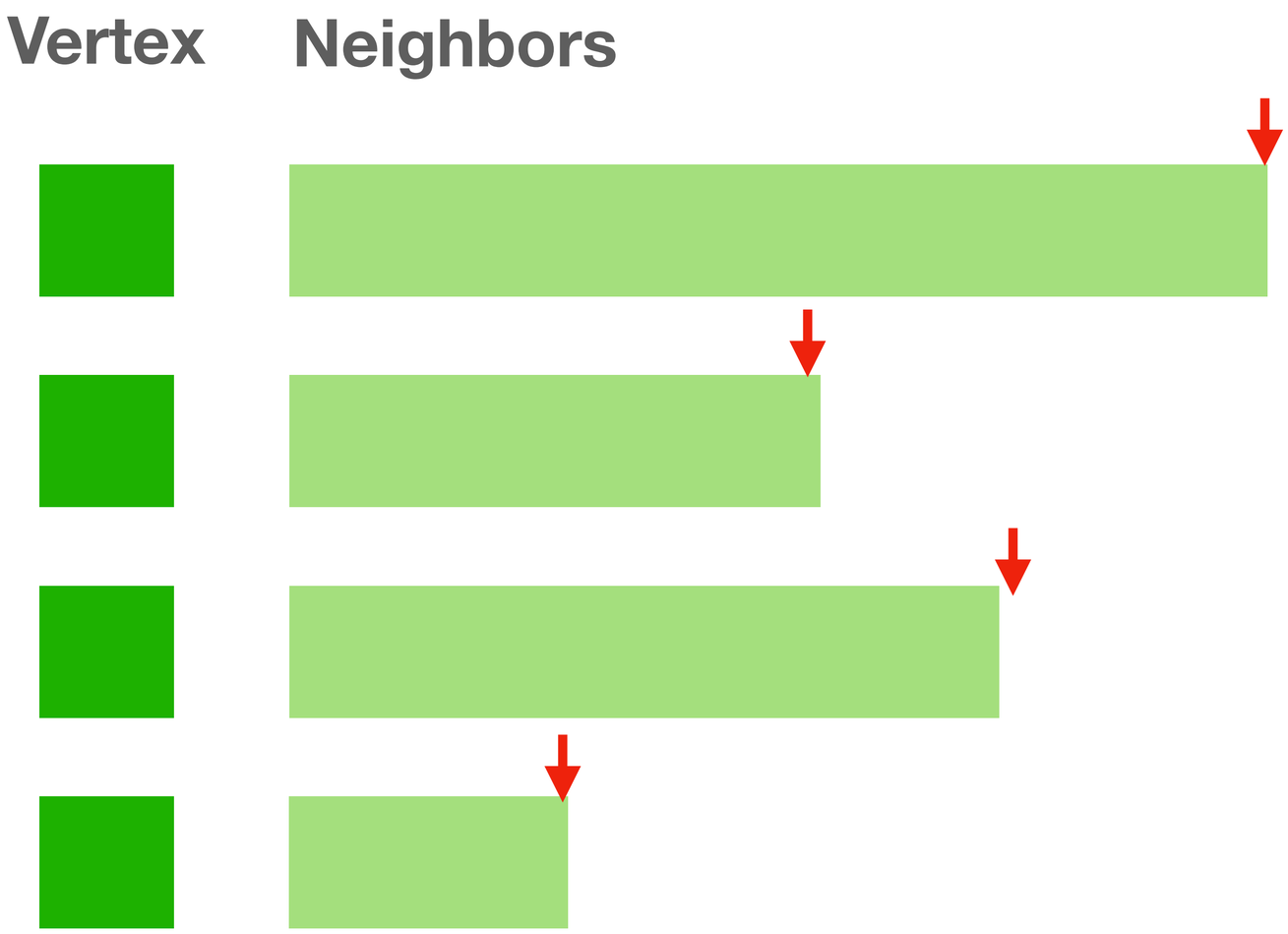
Current approach: Checks all edges in original graph data structure to revert changes

Undo cost **bottlenecks** in dense subproblems



Faster Backtracking

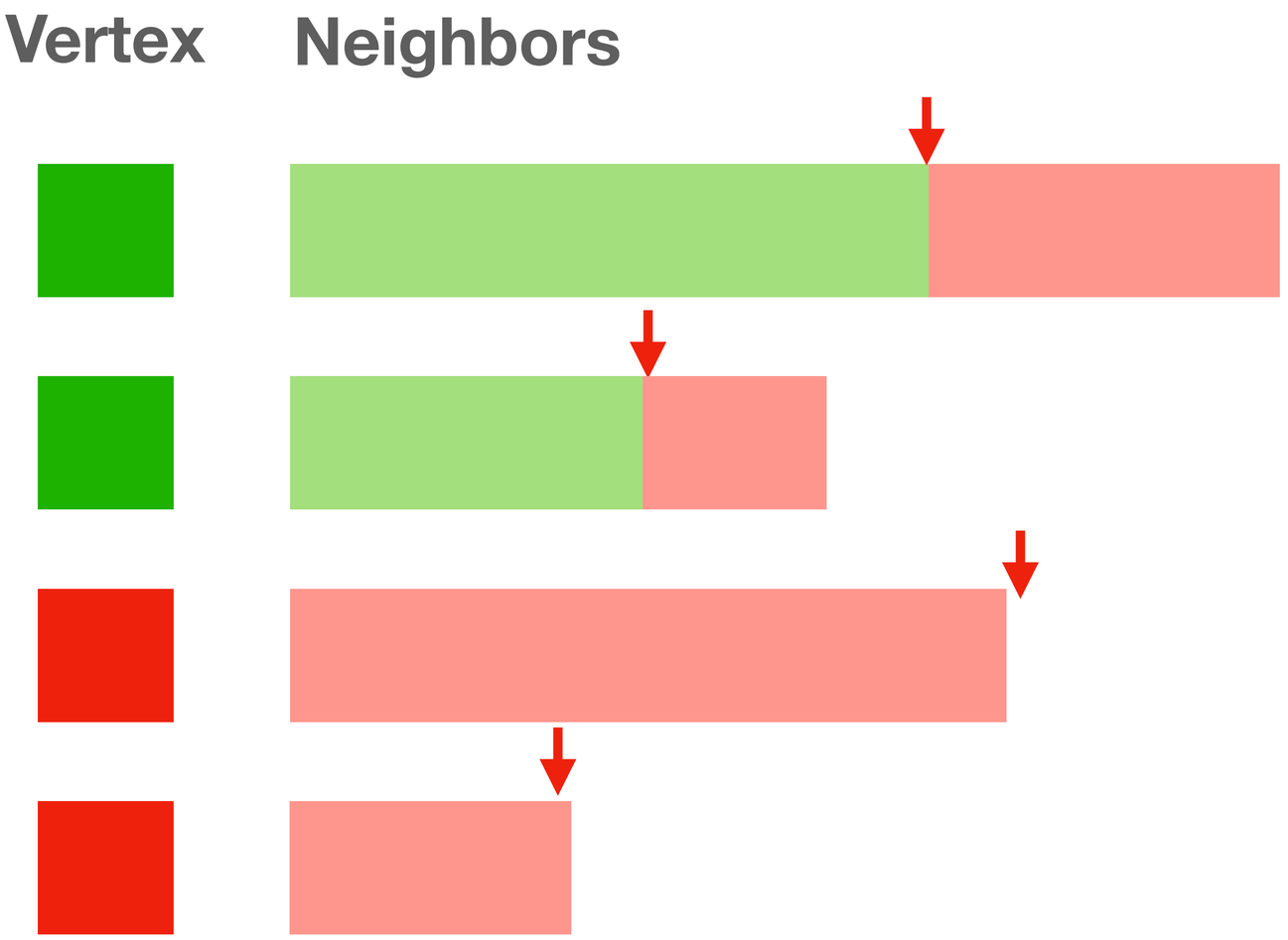
Store:
Dropped vertex



Faster Backtracking

Apply Changes to Graph

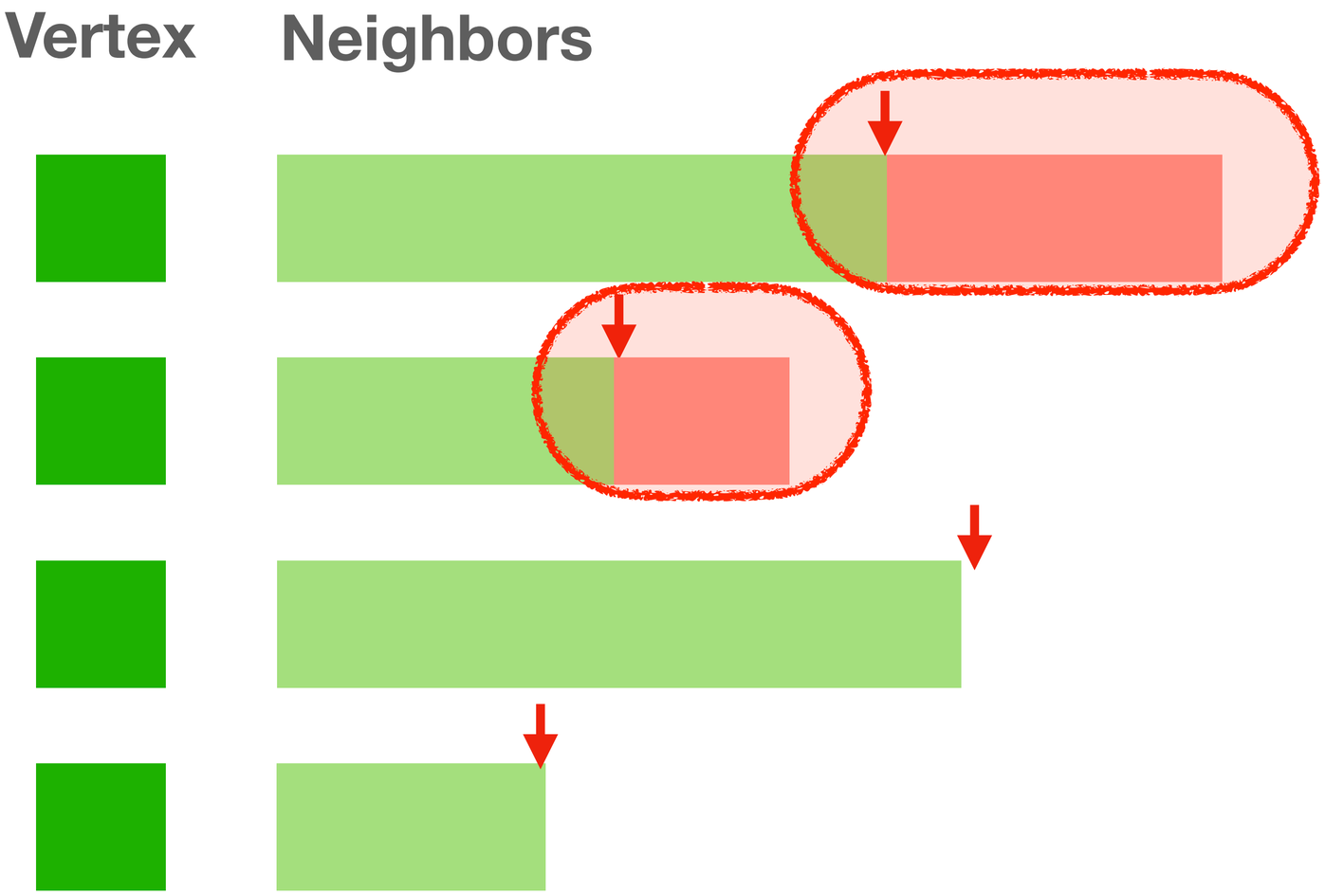
Store:
Dropped vertex



Issue 1: Faster Backtracking

Undo Changes to Graph

Store:
Dropped vertex

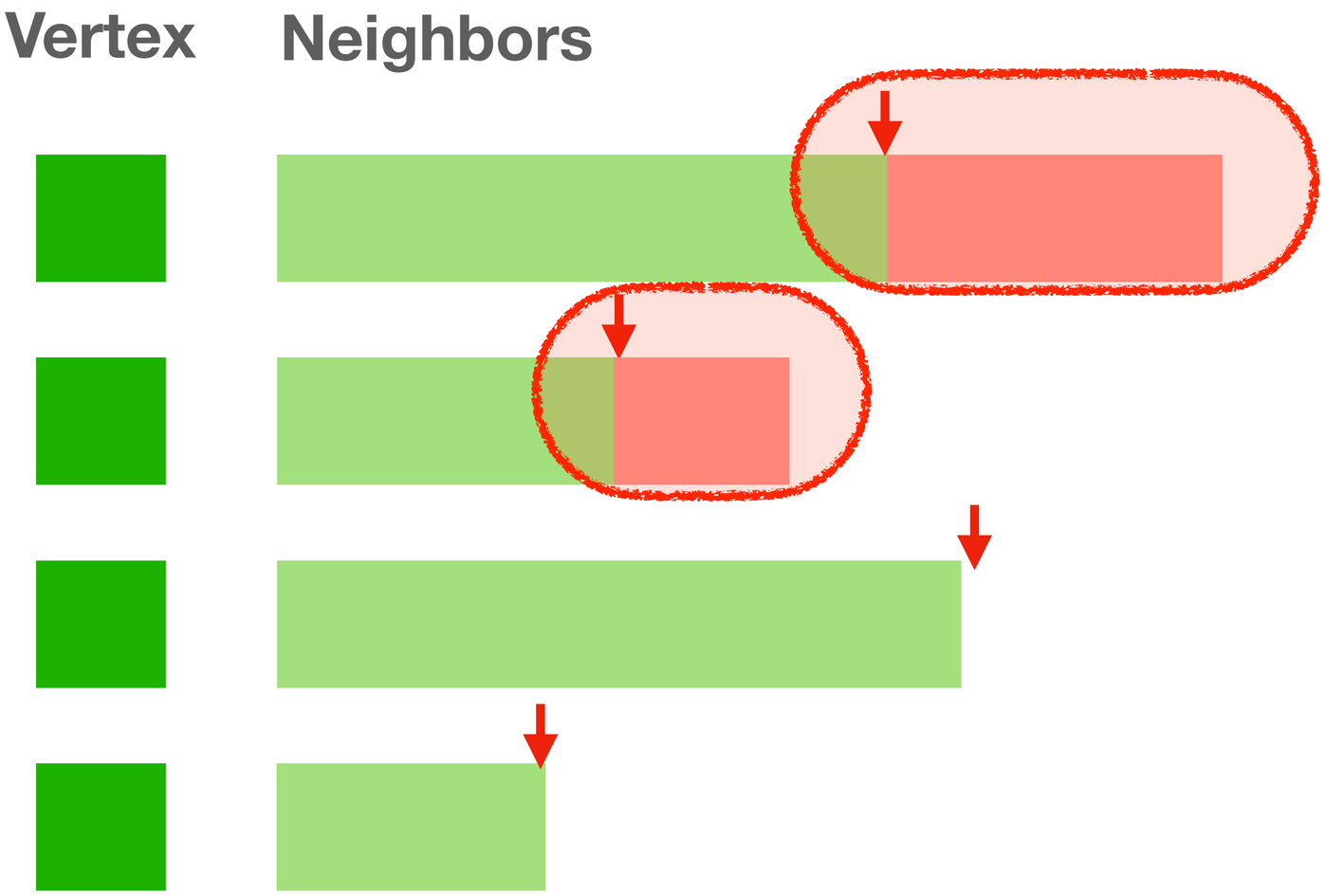


Faster Backtracking

Undo Changes to Graph

Store:
Dropped vertex
& its tail pointer

Active vertex tail

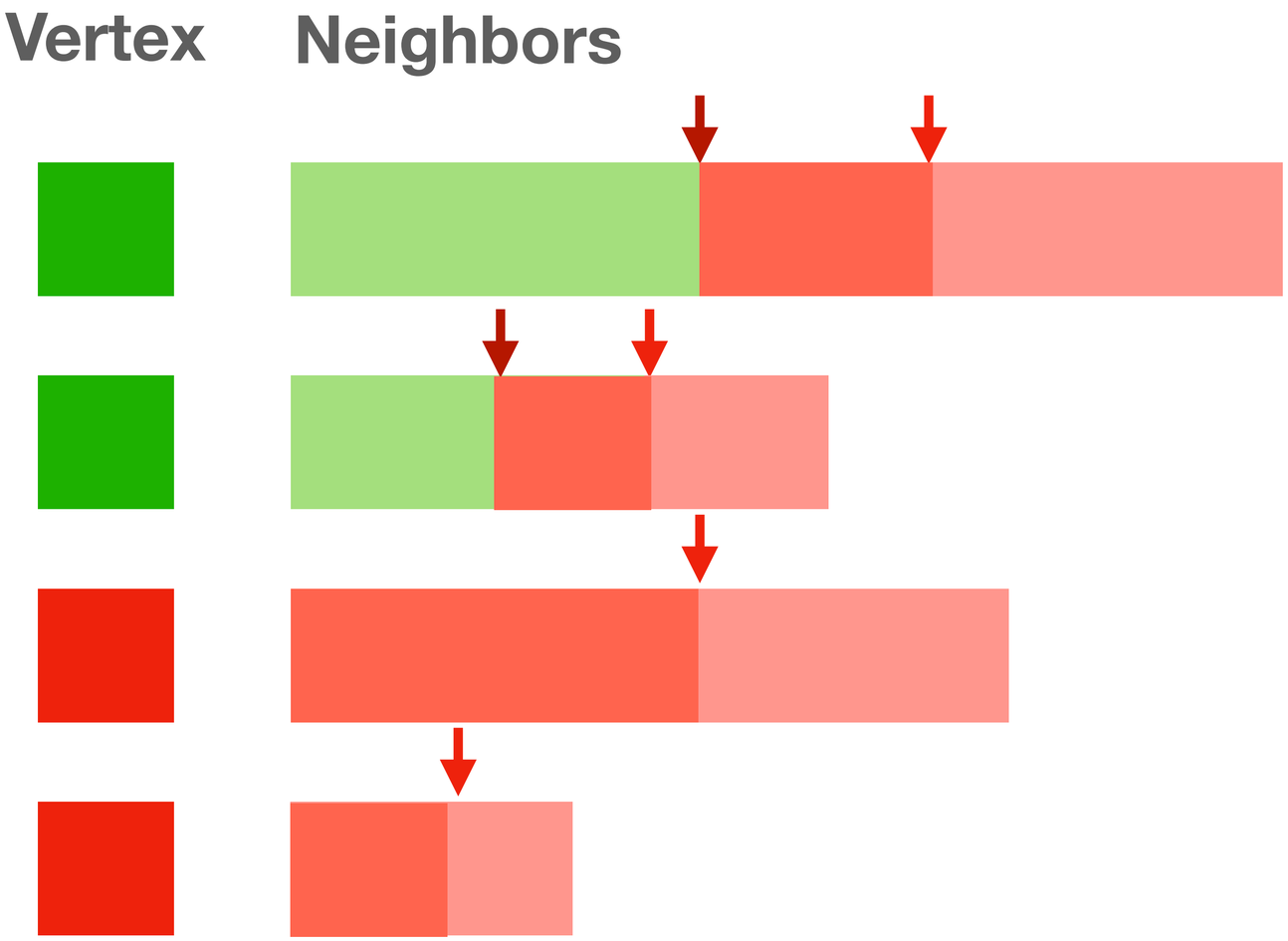


Faster Backtracking

Undo Changes to Graph

Store:
Dropped vertex
& its tail pointer

Active vertex tail

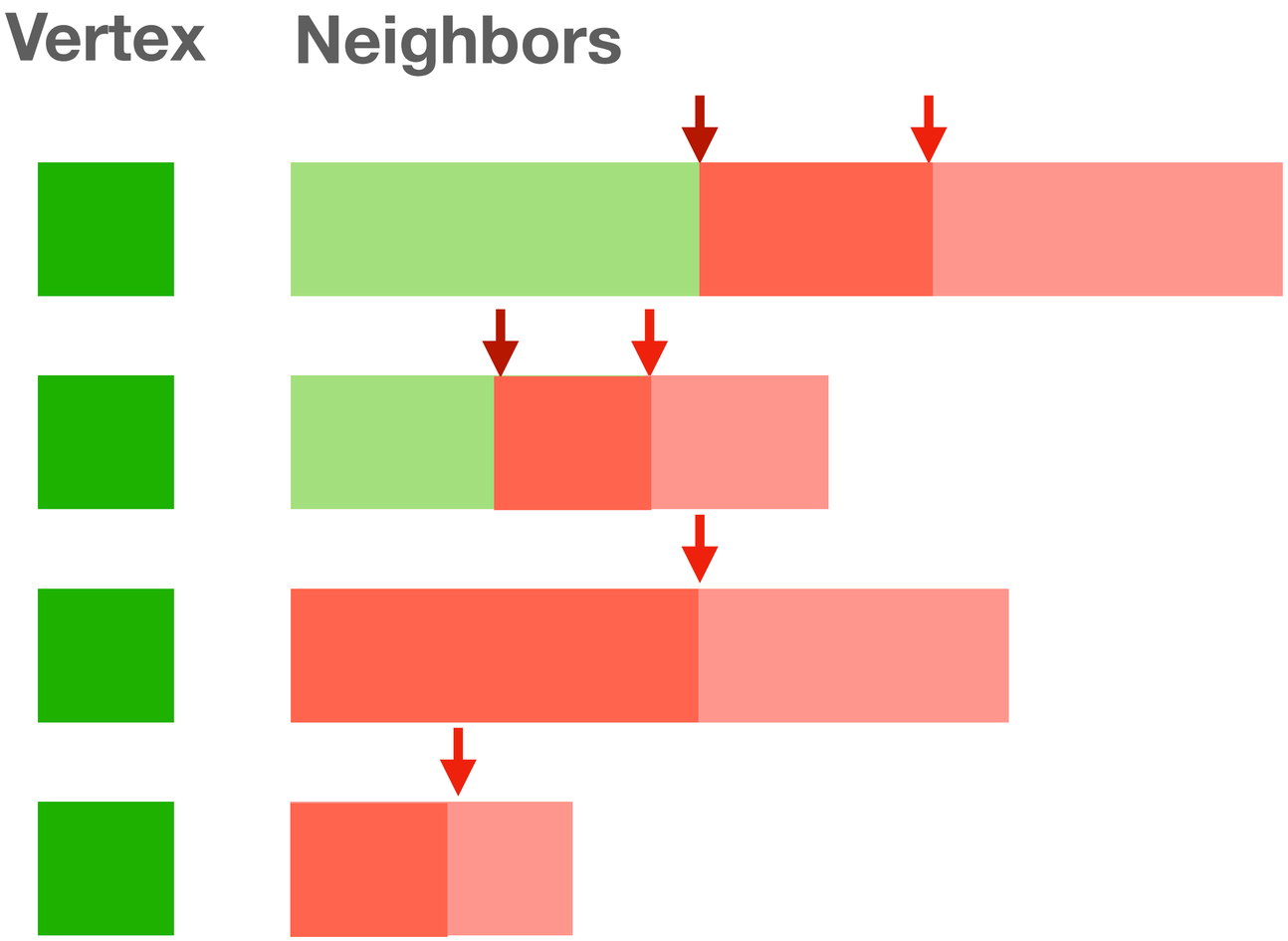


Faster Backtracking

Undo Changes to Graph

Store:
Dropped vertex
& its tail pointer

Active vertex tail



Faster Backtracking

Undo Changes to Graph

Store:
Dropped vertex
& its tail pointer

Active vertex tail

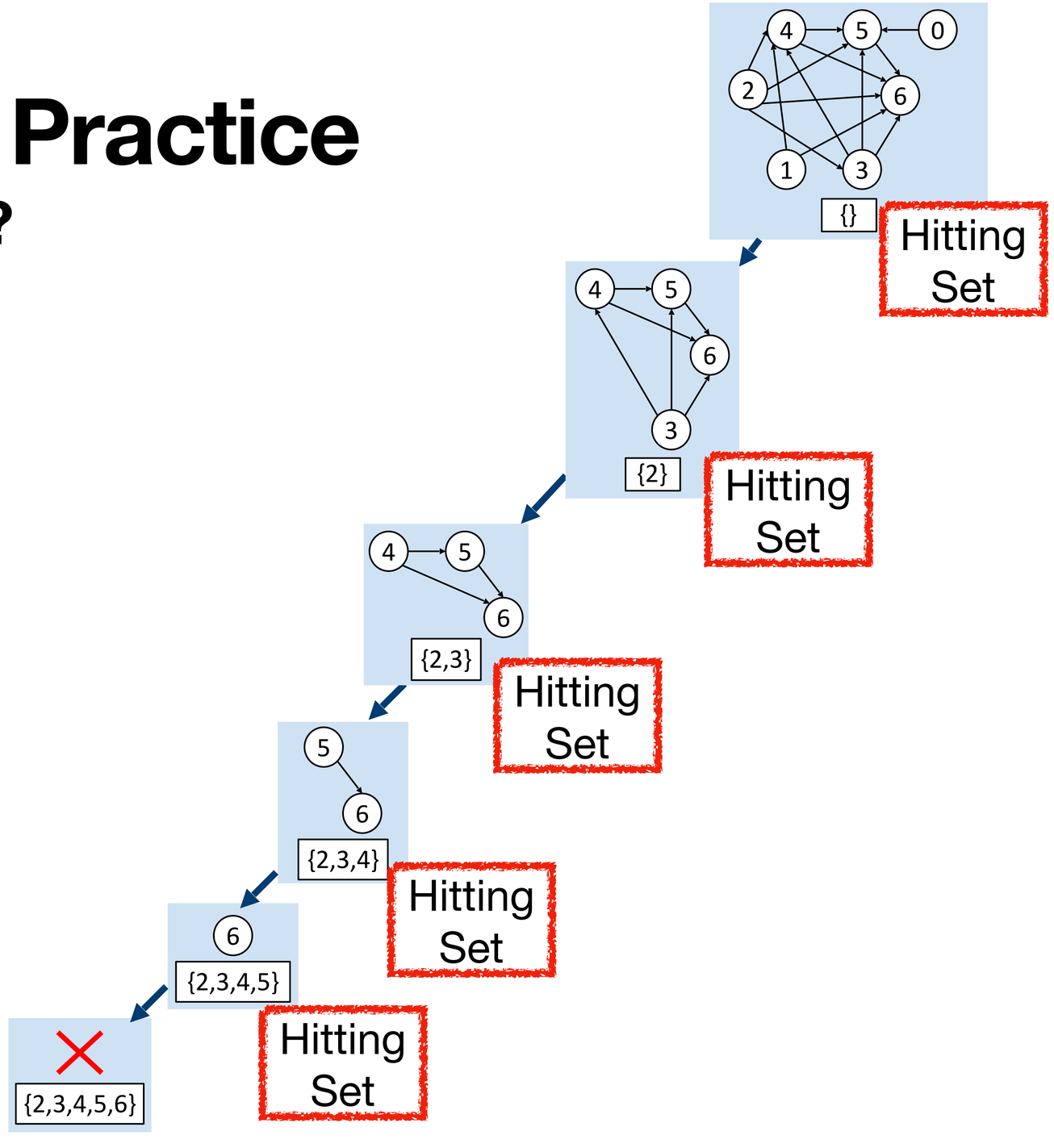


Scaling Challenges in Practice

Issue 2: How to store hitting set?

Hitting set size can be O(|E|)
of each subproblem

Unscalable memory overhead
required to store at
large recursion depth



Scaling Challenges in Practice

Issue 2: How to store hitting set?

Only stores edges (u,v) in the hitting set for each vertex u at a time

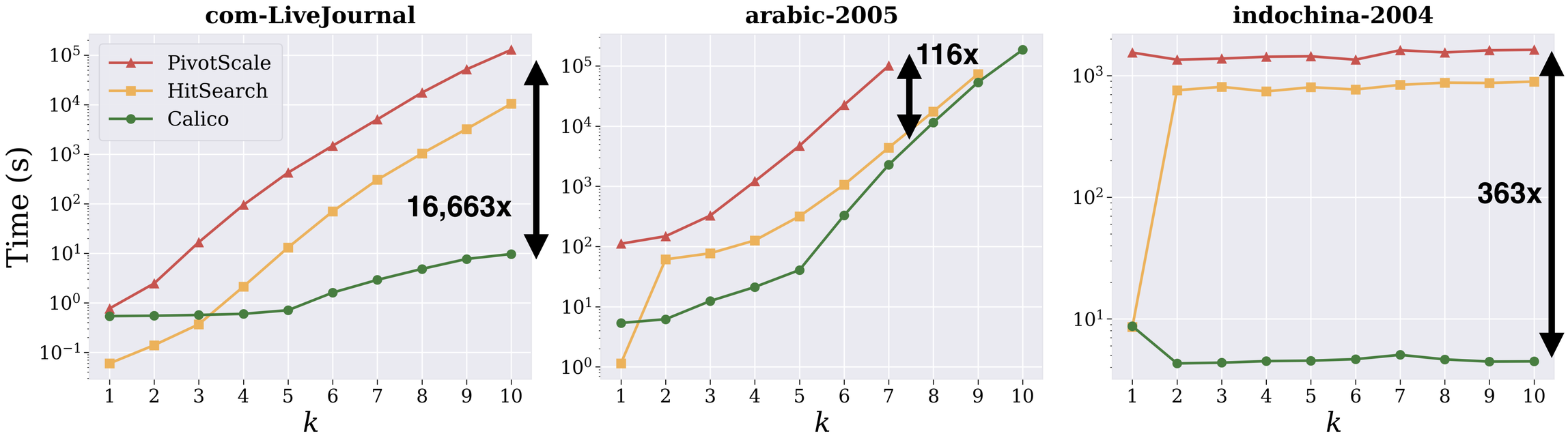
```
if (k − (|C| − |P|) => 3) ∧ (k − |C| >= 3) then
    H ← FindHittingSet(G, p, k − |C|)
    for e = (u, v) ∈ H do
        G_e ← InduceSubgraph(G, e, k − |C|)
        count += Clover(G_e, k, C ∪ {u, v}, P)
```



```
for vertex u in subgraph:
  for vertex v in nbr(u):
    if isEligibleEdge(u,v) :
      Store in HittingSet(u)

  for v in HittingSet(u):
      InduceSubgraph (u,v)
      Branch
```

Better Parallel Performance

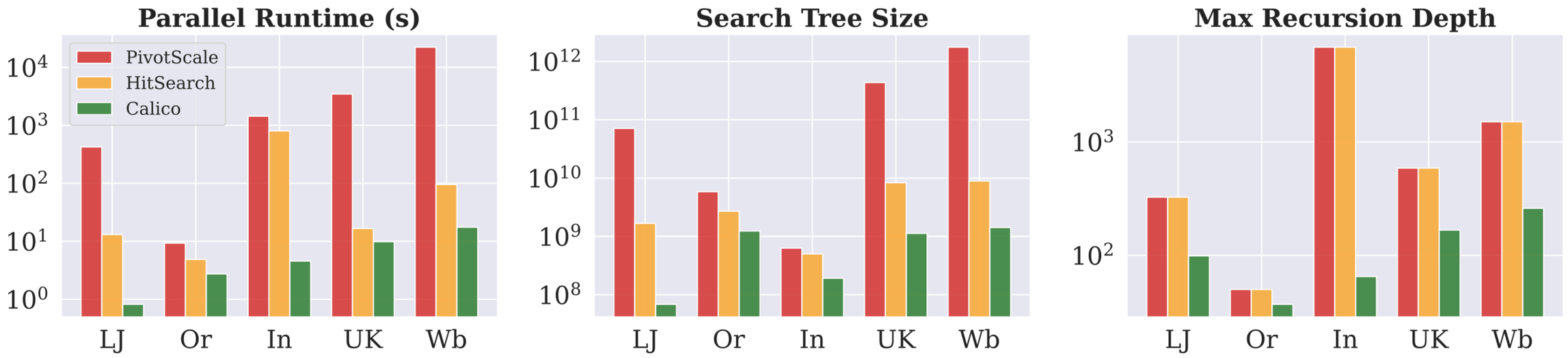


Parameter:
CD = 20
SD = 75

System: Xeon 6972P (Granite Rapids) with 192 cores, 1.5 TiB DDR5 RAM

HitSearch & Calico - This work

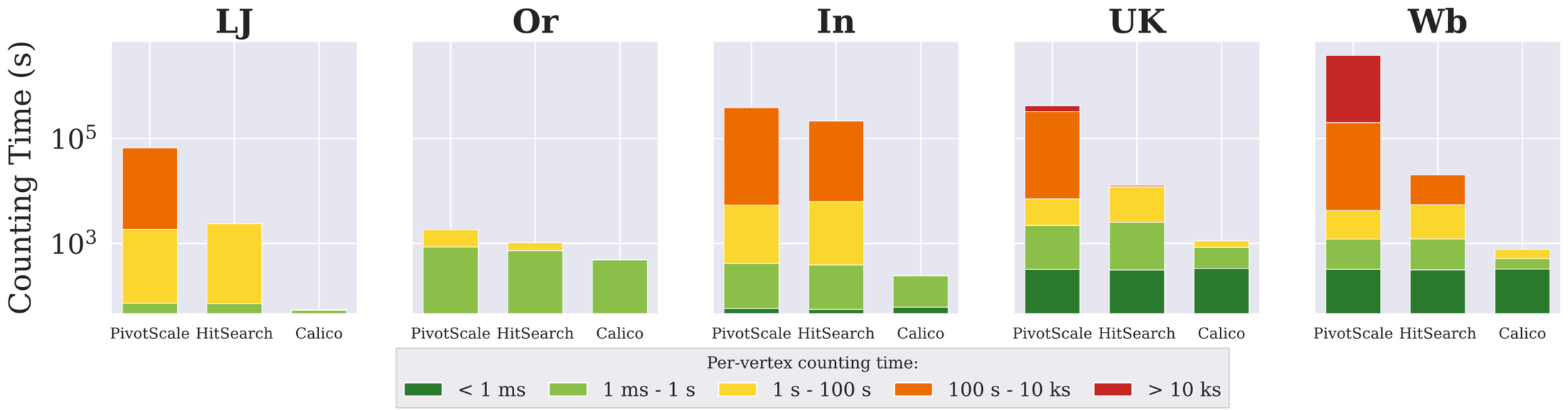
Smaller Search Tree Size & Depth



LJ: com-LiveJournal Or: com-Orkut In: indochina-2004 UK: uk-2005 Wb: webbase-2001

HitSearch & Calico - This work

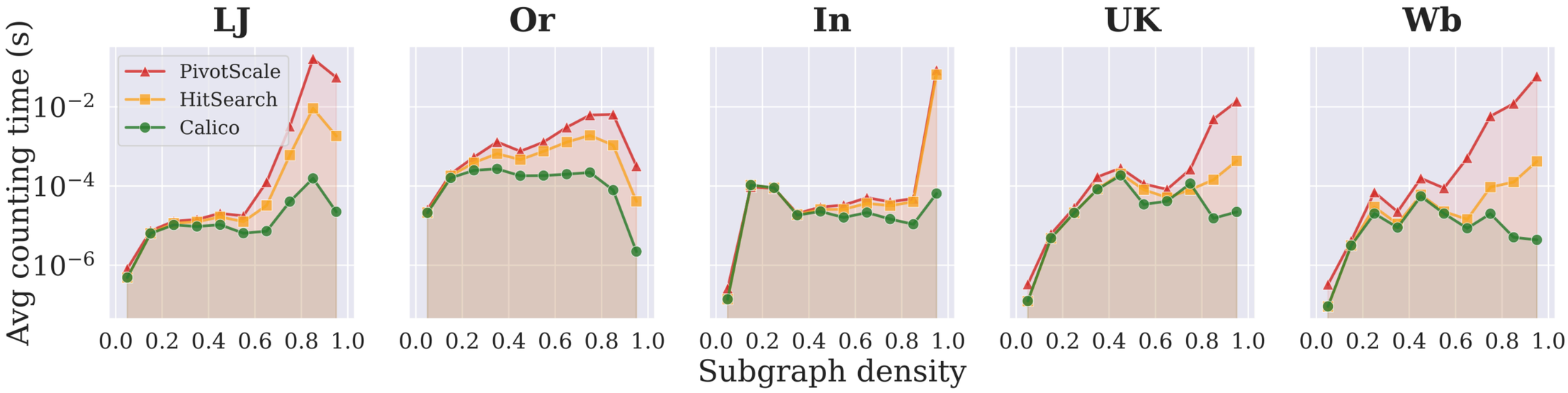
Improved Load Balance



LJ: com-LiveJournal Or: com-Orkut In: indochina-2004 UK: uk-2005 Wb: webbase-2001

HitSearch & Calico - This work

Solve Dense Subproblems Faster



LJ: com-LiveJournal Or: com-Orkut In: indochina-2004 UK: uk-2005 Wb: webbase-2001

HitSearch & Calico - This work

Key Improvements

Avoid Wasted Work

Reduce Sequential Load Imbalance

Lower Memory Overhead

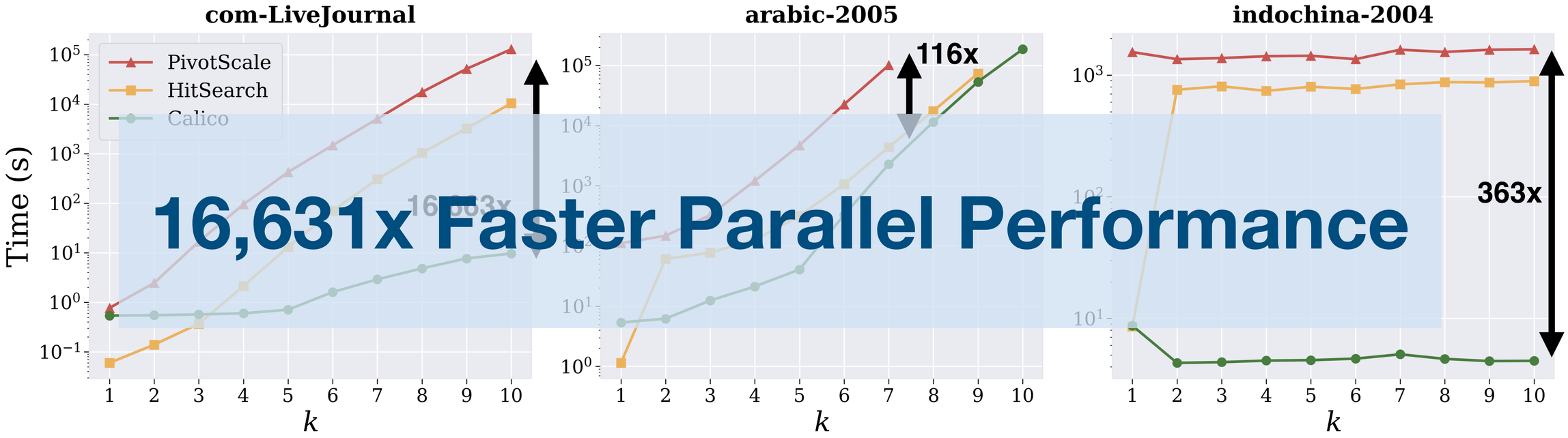
By Addressing

Excess Branching

Recursion Depth

Backtrack Storage

Code available:



16,631x Faster Parallel Performance