# Algorithm to find

- Whether graph has clique of given size
- Maximum clique size
- Number of cliques having maximum size
- Enumerate all cliques having maximum size

# Algorithm is

- constructive
- usable
- complete exhaustive search
- $n = |V|$
  - better than quasi-polynomial time
    $f(n) < \mathcal{O}(n^{log(n)})$ where $n$ is up to few 1000($s$).
  - slower than quasi-polynomial time when $n$ is above few 1000($s$).
- polynomial space complexity - $\mathcal{O}(n^3)$ where $n = |V|$

# Relationship between Clique & Partition

- Each vertex in a clique comes from different partition.
- Clique of size $K$ has vertices from $K$ different partition.
- Number of partitions in a graph can be greater than or equal to maximum clique size of the graph.

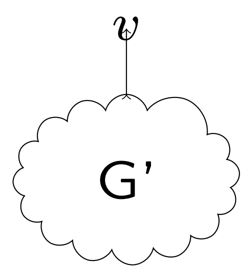
# Maximum clique size of $G$ vs Maximum clique size for vertex $v$

- Let $G$ be a graph.
- Maximum clique size of $G$ is maximum of maximum clique size of any vertex in $G$.

## Maximum clique size for vertex $v$ with respect to its neighborhood graph

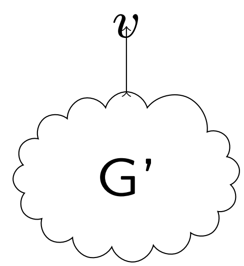
- Let $G$ be a graph and $v$ be one of its vertices.
- Let $G'$ be neighborhood graph of $v$ in $G$.
- Maximum size of any clique having $v$ is $1 +$ maximum clique size of $G'$.

# Neighborhood graph



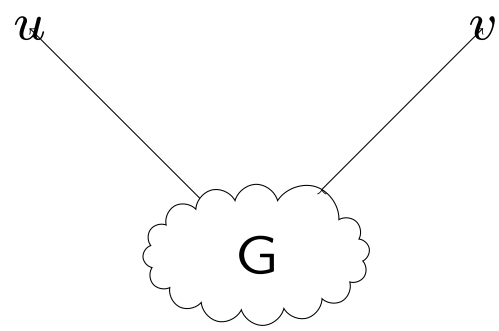

- Let $G'$ be a graph having maximum clique size $K$
- Create graph $G$ by adding vertex $v$ to $G'$.
- If $v$ is connected to all vertices of $G'$ then maximum clique size of $G$ is $K+1$.
- In other words, $G'$ being neighborhood graph of vertex $v$ of graph $G$

# Neighborhood graph



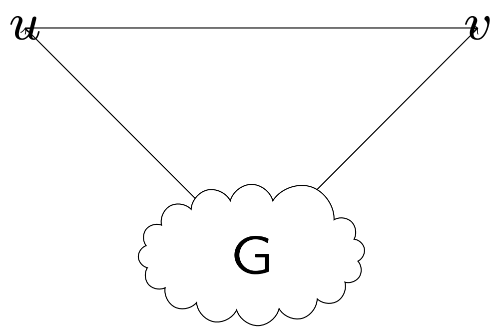

- Let $G$ be a graph.
- Let vertex $v$ is part of $G$.
- Let $K$ be the maximum clique size of vertex $v$.
- Let $G'$ be the neighborhood graph of $v$.
- Maximum clique size of $G'$ is $K - 1$.
- If $v$ is connected to every other vertex in $G$ then $G'$ is same as $G - v$.
- Once maximum clique size of $v$ is found then we can stop processing further.

# Neighborhood graph



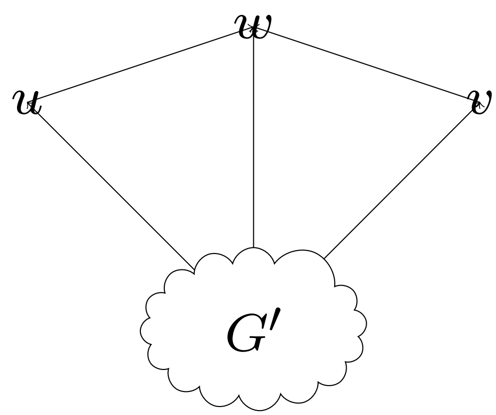

- Let $G$ be a graph having maximum clique size $K$
- Create graph $G^1$ by adding vertex $u$ to $G$ such that $u$ is connected to all vertices in $G$.
- Create graph $G^2$ by adding vertex $v$ to $G$ such that $v$ is connected to all vertices in $G$.
- Maximum clique size of $G^1$ is $K + 1$
- Maximum clique size of $G^2$ is $K + 1$
- Maximum clique size of $G^1$ is equal to maximum clique size of $G^2$
- In other words, if neighborhood graph of vertices $u$ and $v$ are same then their maximum clique size is also same.
- If vertices $u$ and $v$ are part of graph $H$ and if their neighborhood graph is same then we need to find maximum clique size of neighborhood graph of either vertex $u$ or vertex $v$.

# Neighborhood graph




- Let $G$ be a graph having maximum clique size $K$
- Create graph $G'$ by adding vertex $u\&v$ to $G$ such that both $u\&v$ is connected to all vertices in $G$.
- Maximum clique size of $G'$ is $K+2$
- Since $u$ is part of $v$'s neighborhood graph and vice versa, we need to find maximum clique for either one.

# Neighborhood graph




- Let $G$ be a graph.
- Let $u, v\&w$ be one of the vertices of $G$.
- Let $G'$ be the subgraph $G$ without $u, v\&w$.
- Let $u, v\&w$ are connected to every vertices in $G'$
- Let $u, \&v$ are connected to $w$.
- In this case $w$ would be part of neighborhood graph of both $u, \&v$.
- Maximum size of clique having $u$ is same as maximum size of clique having $v$.
- Here, we need to search neighborhood graph of either $u$ or $v$ only. Since $w$ is connected to both $u, \&v$ and thereby part of $u, \&v$'s neighborhood graph, maximum size of clique having $w$ is same as $u$ or $v$.

# Active neighborhood graph

- Let $G$ be a graph.
- Let $V$ be set of vertices of $G$.
- Let $V^1$ be the set of vertices of $V$ which are already searched for the maximum clique.
- Let $V^2$ be the set of vertices of $V$ not in $V^1$. $V^2 = \{V - V^1\}$.
- Members of $V^2$ are not yet searched.
- Let vertex $v$ be one of the vertices of $V^2$.
- Let the neighborhood graph created for vertex $v$ to search be $G^1$.
- It is enough to restrict the members of $G^1$ be the vertices from $V^2$. No need to consider members of $V^1$.
- Now, if there exists a vertex $u$ of $V^1$ whose neighborhood graph is superset of $G^1$ then we don't need to search.

# Recursion depth

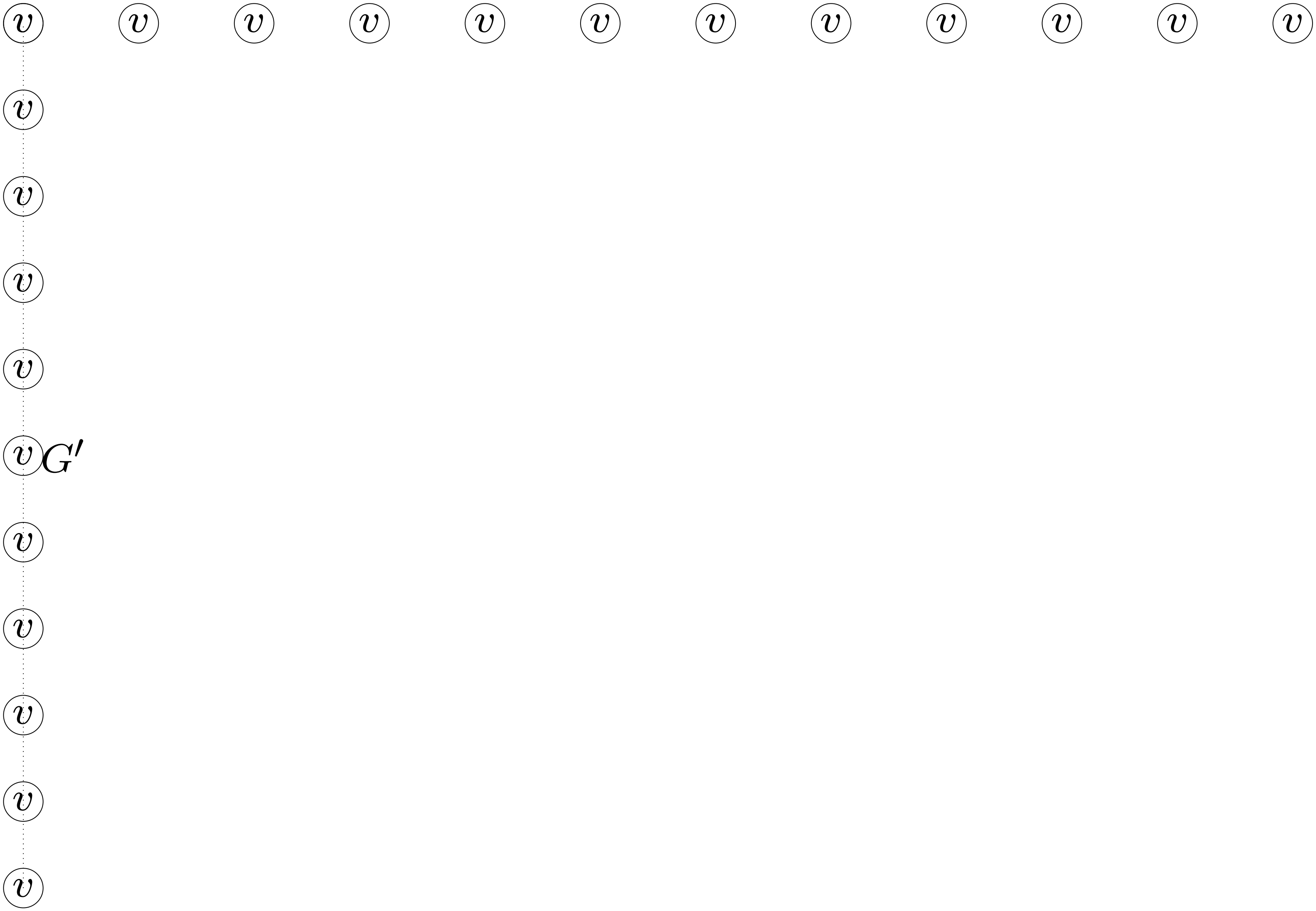
- Recursion depth never exceeds maximum clique size of the given graph $G$ when using neighborhood search.

# Partition extraction

- Let $G$ be a graph.
- Let $v$ be a vertex of $G$.
- If $v$ is connected to every other vertex in this graph $G$, then maximum clique size of $G$ is 1 + maximum clique size of $\{G - v\}$.

# Partition extraction

- Let $G$ be a graph.
- Let $V$ be all vertices of $G$.
- Let $v$ be a vertex of graph $G$.
- Let $V^1$ be the set of vertices to which $v$ is not connected.
- Let $V^2$ be $\{V^1 + v\}$.
- Let $V^3$ be $\{V - V^2\}$.
- Let $G^1$ be induced graph of $G$ containing all vertices of $V^3$ only.
- Now, if vertex $v$ is connected to all vertices in $V^3$ then $G^1$ becomes neighborhood graph of $v$.
- In addition, if all vertices in $V^2$ are independant and forms a partition, then maximum clique size of $G$ is 1 + maximum clique size of $G^1$.
- In this case, we can skip searching neighborhood graph of all vertices of $V^1$.


$v$
$G'$

# Top-down processing
# (Partition Extraction)

- If a partition $P$ can be extracted from active graph $G$'' then extract the partition.
- Increment count of partitions found.
- Reduce active graph $G$'' to $G'' - P$.
- Repeat until there is no more partition to be extracted.

# Bottom-up processing
# (Neighborhood graph duplicate elimination)

Once neighborhood graph search is complete then

- Remove the vertex $v$ whose neighborhood graph is explored from active graph.
- Remove any vertex $u$ part of active graph that has same neighborhood graph as $v$.
- Repeat this process for each vertex $v$ that is removed till there is no more vertex to be removed.

# Creating neighborhood graph of vertex $v$

- if the search is for maximum clique then neighborhood graph is subgraph of $G$ containing list of vertices to which $v$ is connected.
- If the search is for a specific clique size $K$ then neighborhood graph is subgraph of $G$ containing list of vertices to which $v$ is connected and have at least $K-1$ edges.

# Active neighborhood graph search

Ensure that neighborhood graph that needs to be searched is not same or subset of already searched vertex's neighborhood graph

**Algorithm 1** $FindMaximumCliqueSize$ : O(n$^n$)

**function** FindMaximumCliqueSize($G(V, E)$)
    $kMax \leftarrow 0$
    **for all** ($v$ in $V$) **do**
        $G' \leftarrow neighborhood(G, v)$
        $kMax_1 \leftarrow FindMaximumCliqueSize(G')$
        **if** (($kMax_1 + 1) > kMax$)) **then**
            $kMax \leftarrow kMax_1 + 1$
        **end if**
    **end for**
    **return** $kMax$
**end function**

**Algorithm 2** $FindMaximumCliqueSize$ : O(n!)

```
function FindMaximumCliqueSize(G(V, E))
    kMax ← 0
    H ← G
    for all (v in V) do
        G' ← neighborhood(H, v)
        kMax_1 ← FindMaximumCliqueSize(G')
        if ((kMax_1 + 1) > kMax) then
            kMax ← kMax_1 + 1
        end if
        H ← {H − v}
    end for
    return kMax
end function
```

**Algorithm 3** $FindMaximumCliqueSize$ : O(n!)

**function** FindMaximumCliqueSize($G(V, E), k$)<br>
$kMax \leftarrow max(k-1, 0)$<br>
$H \leftarrow G$<br>
**for all** ($v$ in $V$) **do**<br>
$G' \leftarrow neighborhood(H, v, kMax)$<br>
**if** ($|G'| >= kMax$) **then**<br>
$(found_1, kMax_1) \leftarrow FindMaximumCliqueSize(G', kMax)$<br>
**if** ($found_1$ and $((kMax_1 + 1) > kMax)$) **then**<br>
$kMax \leftarrow kMax_1 + 1$<br>
**end if**<br>
**end if**<br>
$H \leftarrow \{H - v\}$<br>
**end for**<br>
**return** $(kMax >= k, kMax)$<br>
**end function**

# Complete multipartite graph

- A complete multipartite graph is a graph that is complete $k$-partite for some $k$.
- A complete $k$-partite graph is a $k$-partite graph in which there is an edge between every pair of vertices from different independent sets.

A complete 6-partite sample graph $K_{2,2,2,2,2,2}$

$$G = \{v_{1:1}, v_{1:2}, v_{2:1}, v_{2:2}, v_{3:1}, v_{3:2}, v_{4:1}, v_{4:2}, v_{5:1}, v_{5:2}, v_{6:1}, v_{6:2}\}$$

$\{v_{1:1}, v_{1:2}\}$

$\{v_{2:1}, v_{2:2}\}$

$\{v_{3:1}, v_{3:2}\}$

$\{v_{4:1}, v_{4:2}\}$

$\{v_{5:1}, v_{5:2}\}$

$\{v_{6:1}, v_{6:2}\}$

- Time complexity : $f(x) = 2^6 + 2^5 + 2^4 +$
- Number of distinct cliques of size 6 is $2^6$
- Space to store $2^6$ 6-clique is $|V|$

A complete 6-partite sample graph $K_{3,3,3,3,3,3}$

$$G = \{v_{1:1}, v_{1:2}, v_{1:3}, v_{2:1}, v_{2:2}, v_{2:3}, v_{3:1}, v_{3:2}, v_{3:3}, v_{4:1}, v_{4:2}, v_{4:3}, v_{5:1}, v_{5:2}, v_{5:3}, v_{6:1}, v_{6:2}, v_{6:3}\}$$

$\{v_{1:1}, v_{1:2}, v_{1:3}\}$

$\{v_{2:1}, v_{2:2}, v_{2:3}\}$

$\{v_{3:1}, v_{3:2}, v_{3:3}\}$

$\{v_{4:1}, v_{4:2}, v_{4:3}\}$

$\{v_{5:1}, v_{5:2}, v_{5:3}\}$

$\{v_{6:1}, v_{6:2}, v_{6:3}\}$

- Time complexity : $f(x) = 3^6 + 3^5 + 3^4 + 3^3 +$
- Number of distinct cliques of size 6 is $3^6$
- Space to store $3^6$ 6-clique is $|V|$

**Algorithm 4** $FindMaximumCliqueSize$ : $\mathsf{O}(\mathsf{exp}^{(\mathsf{f(n)})})$

**function** FindMaximumCliqueSize($G(V, E), k$)
    $kMax \leftarrow max(k - 1, 0)$
    $partitions \leftarrow \{\}$
    $H \leftarrow G$
    **while** $(|H| > kMax)$ **do**
        **while** $|(p := ExtractPartition(H))| > 0$ **do**
            $partitions \leftarrow partitions \cup p$
            $H \leftarrow H - p$
            $kMax \leftarrow max(kMax - 1, 0)$
        **end while**
        **if** $(H = \{\})$ **then**
            **break**
        **end if**
        $v \leftarrow PickAVertex(H)$
        $G' \leftarrow neighborhood(H, v, kMax)$

**if** $(|G'| >= kMax)$ **then**<br>
  $(found_1, kMax_1) \leftarrow FindMaximumCliqueSize(G', kMax)$<br>
  **if** $(found_1$ and $((kMax_1 + 1) > kMax))$ **then**<br>
    $kMax \leftarrow kMax_1 + 1$<br>
  **end if**<br>
**end if**<br>
$H \leftarrow \{H - v\}$<br>
**end while**<br>
$kMax \leftarrow |partitions| + kMax$<br>
**return** $(kMax >= k, kMax)$<br>
**end function**

# Top-down

- The above algorithm uses top-down optimization.
- Time complexity is $O(n^3)$ for complete multipartite graphs.
- Still not good enough for random graph.

**Algorithm 5** $FindMaximumCliqueSize$ : O(exp$^{(\mathsf{f(n)})}$)

**function** FindMaximumCliqueSize($G(V, E), k$)
  $kMax \leftarrow max(k-1, 0)$
  $H \leftarrow G$
  **while** $(|H| > kMax)$ **do**
    $v \leftarrow PickAVertex(H)$
    $G' \leftarrow neighborhood(H, v, kMax)$
    **if** $(|G'| >= kMax)$ **then**
      $(found_1, kMax_1) \leftarrow FindMaximumCliqueSize(G', kMax)$
      **if** ($found_1$ and $((kMax_1 + 1) > kMax)$) **then**
        $kMax \leftarrow kMax_1 + 1$
      **end if**
    **end if**

```
    G' ← neighborhood(H, v)
    H ← {H − v}
    for all v' ∈ H do
        G'' ← neighborhood(H, v')
        if (isSubgraph(G', G'') then
            H ← {H − v'}
        end if
    end for
    end while
    return (kMax >= k, kMax)
end function
```

**Algorithm 6** $FindMaximumCliqueSize$ : $O(exp^{(f(n))})$

```
function FindMaximumCliqueSize(G(V, E), k)
    kMax ← max(k − 1, 0)
    H ← G
    while (|H| > kMax) do
        v ← PickAVertex(H)
        G′ ← neighborhood(H, v, kMax)
        if (|G′| >= kMax) then
            search ← true
            for all v′ ∈ {G − H} do
                G″ ← neighborhood(G, v′)
                if (isSubgraph(G″, G′)) then
                    search ← false
                end if
            end for
            if search then
                (found_1, kMax_1) ← FindMaximumCliqueSize(G′, kMax)
```

**if** $(found_1$ and $((kMax_1 + 1) > kMax))$ **then**

$kMax \leftarrow kMax_1 + 1$

**end if**

**end if**

**end if**

$G' \leftarrow neighborhood(H, v)$

$H \leftarrow \{H - v\}$

**for all** $v' \in H$ **do**

$G'' \leftarrow neighborhood(H, v')$

**if** $(isSubgraph(G', G'')$ **then**

$H \leftarrow \{H - v'\}$

**end if**

**end for**

**end while**

**return** $(kMax >= k, kMax)$

**end function**

# Bottom-up

- The above algorithm uses bottom-up optimization.
- Time complexity is $O(n^3)$ for complete multipartite graphs.
- Still not good enough for random graph.

**Algorithm 7** $FindMaximumCliqueSize$ : $\mathrm{O}(\mathrm{exp}^{(\mathrm{f(n)})})$

**function** FindMaximumCliqueSize($G(V, E), k$)
    $kMax \leftarrow max(k - 1, 0)$
    $partitions \leftarrow \{\}$
    $H \leftarrow G$
    **while** $(|H| > kMax)$ **do**
        **while** $|(p := ExtractPartition(H))| > 0$ **do**
            $partitions \leftarrow partitions \cup p$
            $H \leftarrow H - p$
            $kMax \leftarrow max(kMax - 1, 0)$
        **end while**
        **if** $(H = \{\})$ **then**
            **break**
        **end if**
        $v \leftarrow PickAVertex(H)$
        $G' \leftarrow neighborhood(H, v, kMax)$
        **if** $(|G'| >= kMax)$ **then**
            $search \leftarrow true$

**for all** $v' \in \{G - H\}$ **do**
$G'' \leftarrow neighborhood(G, v')$
**if** $(isSubgraph(G'', G')$ **then**
$search \leftarrow false$
**end if**
**end for**
**if** $search$ **then**
$(found_1, kMax_1) \leftarrow FindMaximumCliqueSize(G', kMax)$
**if** $(found_1$ and $((kMax_1 + 1) > kMax))$ **then**
$kMax \leftarrow kMax_1 + 1$
**end if**
**end if**
**end if**
$G' \leftarrow neighborhood(H, v)$
$H \leftarrow \{H - v\}$
**for all** $v' \in H$ **do**
$G'' \leftarrow neighborhood(H, v')$
**if** $(isSubgraph(G', G'')$ **then**
$H \leftarrow \{H - v'\}$
**end if**

**end for**

**end while**

$kMax \leftarrow |partitions| + kMax$

**return** $(kMax >= k, kMax)$

**end function**

# Top-down with Bottom-up

- The above algorithm uses top-down with bottom-up optimization.
- Time complexity is $O(n^3)$ for complete multipartite graphs.
- Still not good enough for random graph.

## Vertex selection (PickAVertex)

- Picking a vertex with largest degree.
- Picking a vertex with least degree.

**Algorithm 8** $FindMaximumCliqueSize$ : $\mathsf{O}(\mathsf{exp}^{(\mathsf{f}(\mathsf{n}))})$

```
function FindMaximumCliqueSize(G(V, E), k)
    kMax ← max(k − 1, 0)
    partitions ← {}
    H ← G
    while (|H| > kMax) do
        while |(p := ExtractPartition(H))| > 0 do
            partitions ← partitions ∪ p
            H ← H − p
            kMax ← max(kMax − 1, 0)
        end while
        if (H = {}) then
            break
        end if
        v ← PickAVertexWithLargestDegree(H)
        G' ← neighborhood(H, v, kMax)
        if (|G'| >= kMax) then
            search ← true
            for all v' ∈ {G − H} do
```

```
                G'' ← neighborhood(G, v')
                if (isSubgraph(G'', G')) then
                    search ← false
                end if
            end for
            if search then
                (found_1, kMax_1) ← FindMaximumCliqueSize(G', kMax)
                if (found_1 and ((kMax_1 + 1) > kMax)) then
                    kMax ← kMax_1 + 1
                end if
            end if
        end if
        G' ← neighborhood(H, v)
        H ← {H − v}
        for all v' ∈ H do
            G'' ← neighborhood(H, v')
            if (isSubgraph(G', G'')) then
                H ← {H − v'}
            end if
        end for
```

**end while**

$kMax \leftarrow |partitions| + kMax$

**return** $(kMax >= k, kMax)$

**end function**

# Top-down with Bottom-up: picking vertex with largest degree

- Time complexity is $O(exp^{(\mathsf{f(n)})})$ for random graphs.
- Time complexity for picking random vertex is worst case time complexity of picking either largest degree or least degree vertex for search. In this case, picking a vertex with largest degree.

**Algorithm 9** $FindMaximumCliqueSize$ : $\mathrm{O}(\mathrm{n}^{(\log(\mathrm{n}))})$

**function** FindMaximumCliqueSize$(G(V, E), k)$<br>
    $kMax \leftarrow max(k - 1, 0)$<br>
    $partitions \leftarrow \{\}$<br>
    $H \leftarrow G$<br>
    **while** $(|H| > kMax)$ **do**<br>
        **while** $|(p := ExtractPartition(H))| > 0$ **do**<br>
            $partitions \leftarrow partitions \cup p$<br>
            $H \leftarrow H - p$<br>
            $kMax \leftarrow max(kMax - 1, 0)$<br>
        **end while**<br>
        **if** $(H = \{\})$ **then**<br>
            **break**<br>
        **end if**<br>
        $v \leftarrow PickAVertexWithLeastDegree(H)$<br>
        $G' \leftarrow neighborhood(H, v, kMax)$<br>
        **if** $(|G'| >= kMax)$ **then**<br>
            $search \leftarrow true$<br>
            **for all** $v' \in \{G - H\}$ **do**

```
            G'' ← neighborhood(G, v')
            if (isSubgraph(G'', G')) then
                search ← false
            end if
        end for
        if search then
            (found_1, kMax_1) ← FindMaximumCliqueSize(G', kMax)
            if (found_1 and ((kMax_1 + 1) > kMax)) then
                kMax ← kMax_1 + 1
            end if
        end if
    end if
    G' ← neighborhood(H, v)
    H ← {H − v}
    for all v' ∈ H do
        G'' ← neighborhood(H, v')
        if (isSubgraph(G', G'')) then
            H ← {H − v'}
        end if
    end for
```

**end while**

$kMax \leftarrow |partitions| + kMax$

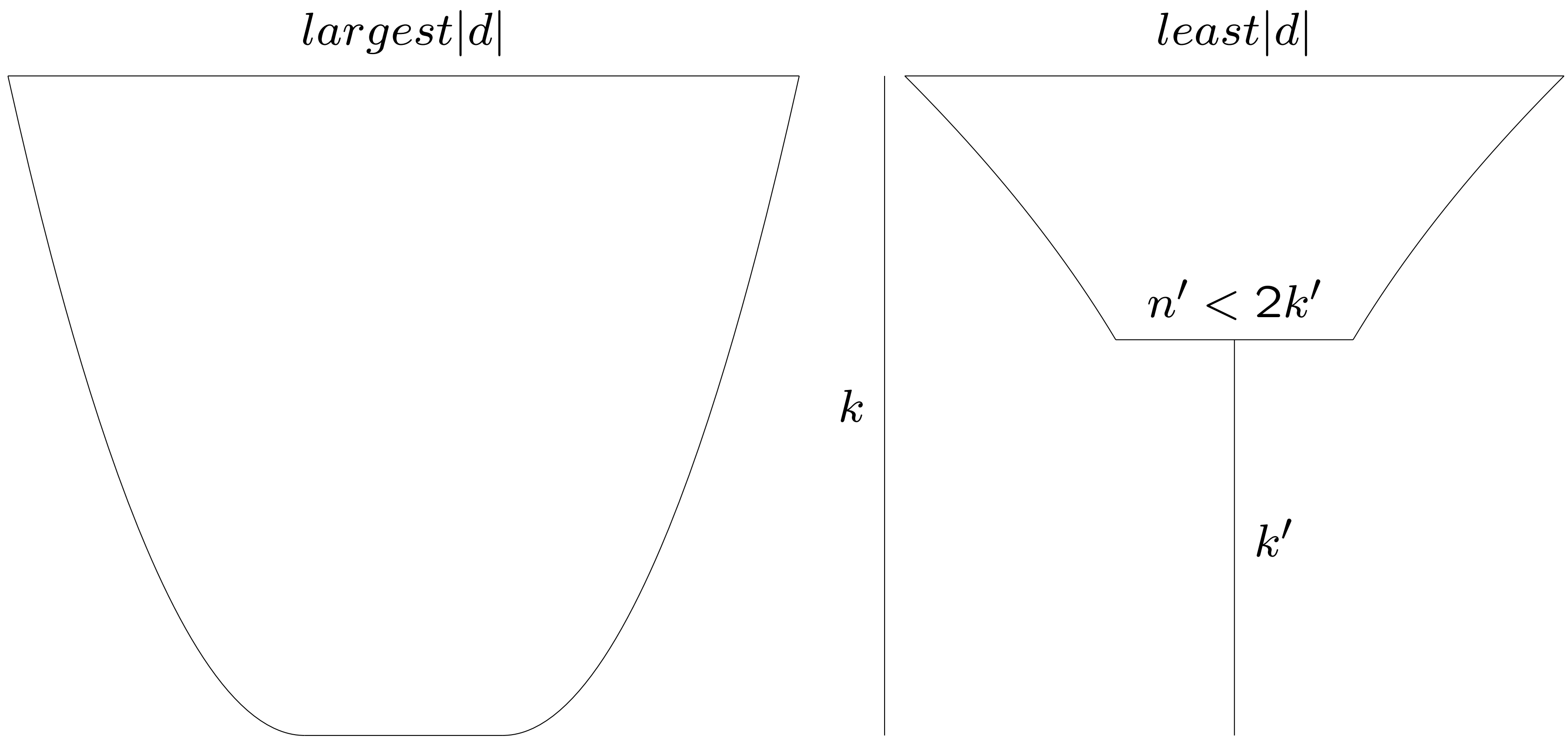
**return** $(kMax >= k, kMax)$

**end function**

# Top-down with Bottom-up: picking vertex with least degree

- Time complexity is better than $O(n^{\mathsf{log}^{\mathsf{n}}})$ for random graphs with $n$ up to few 1000(s). Slower than this when $n$ is above few 1000(s).

WHY?




- $k$ is maximum clique size of graph $G$.
- Shape (horizontal) denotes neighborhood graph size at each recursion level.

# Hot spots

- Checking each neighbor vertex has at least clique-size common neighbors.
- Neighborhood graph generation
- Working set involved in bottom-up (collapsing identical processed neighborhood graph) processing.
- Recursion depth
- Graph size at each recursion level.

# Picking vertex with largest degree

- Convergence of vertices in neighborhood graph at each level to searching clique size is slow.

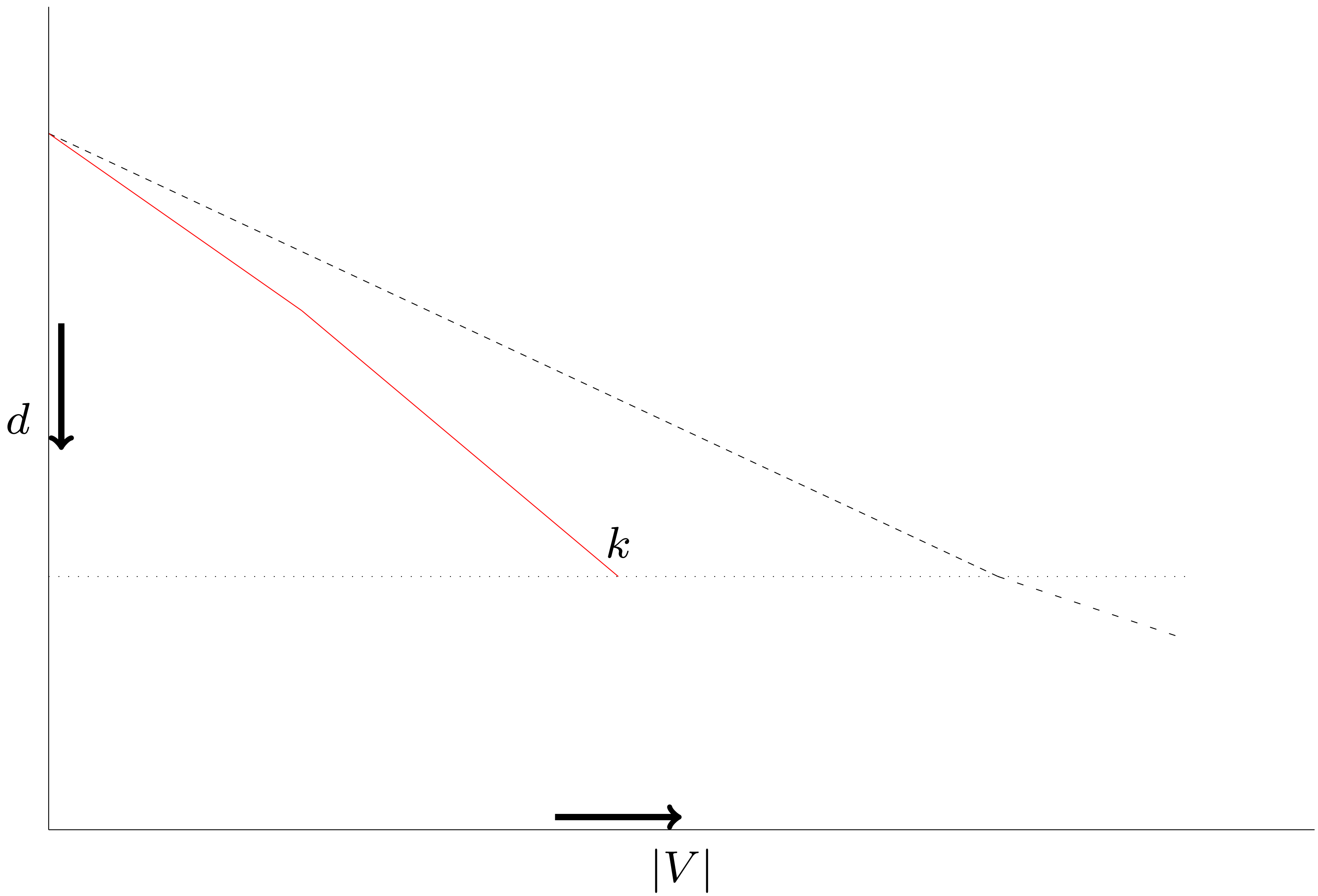
- Vertices to be explored at each level is maximum possible.

- Depth of the recursion is almost equal to maximum clique size of the graph.

- Hit ratio of top-down (partition extraction) and bottom-up (collapse of identical neighborhood graph) are moderate.

- Bottom-up processing needs to process larger set which leads to higher time consumption.

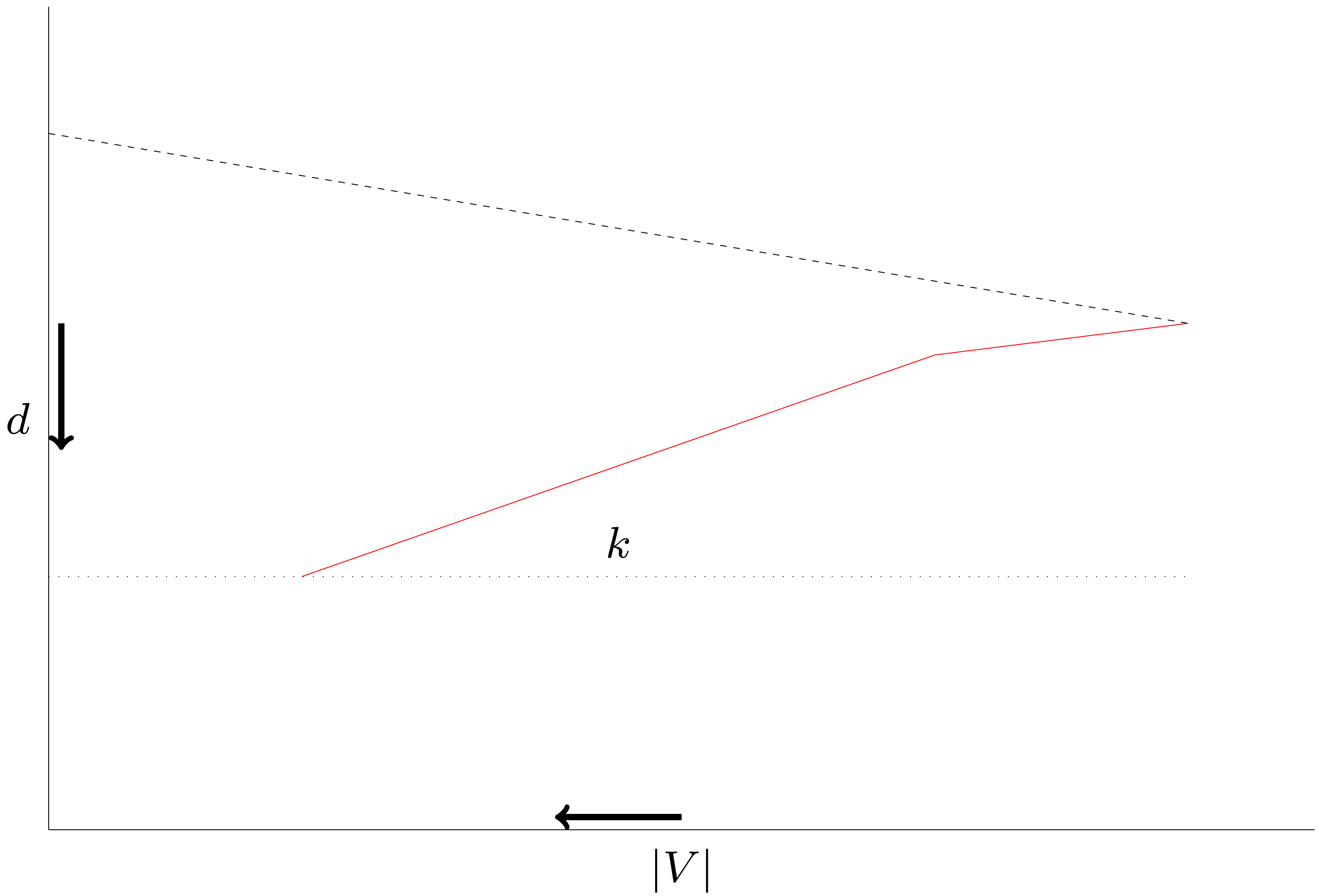
- Pretty much all hot spots are affected negatively to the maximum.



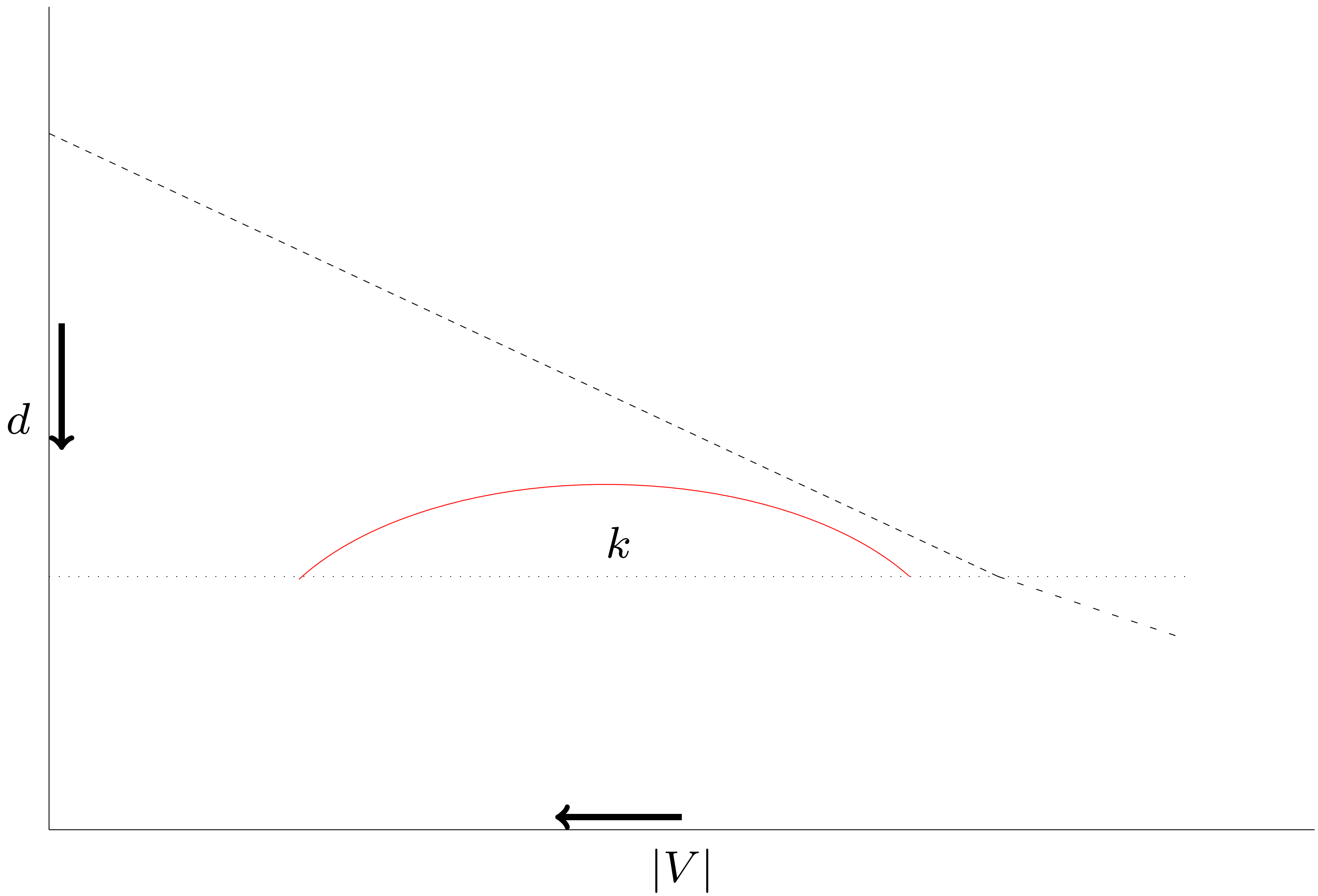
$d$
$k$
$|V|$

# Picking vertex with least degree

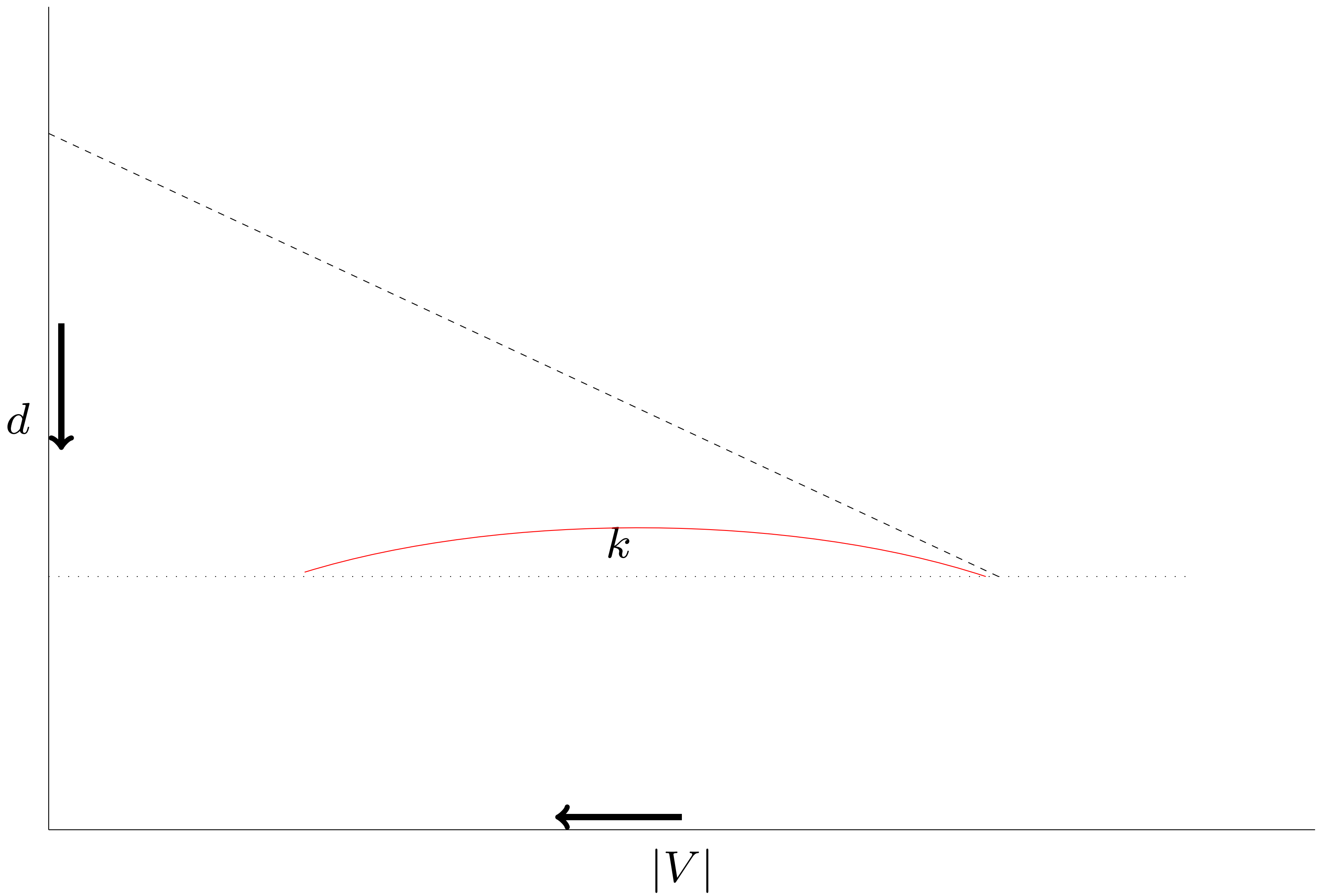
- Convergence of vertices in neighborhood graph at each level to searching clique size is aggressive.
- Iterations at each level is as small as possible.
- Vertices which does not have enough neighbors to produce clique of given size are removed without further processing.
- Hit ratio of top-down (partition extraction) and bottom-up (collapse of identical neighborhood graph) are aggressive.
- Bottom-up processing needs to process smaller set which leads to lower time consumption.
- Quicker convergence leads to denser neighborhood graph at inner recursion levels. This lets top-down (partition extraction) processing possible. Since every partition is a level, every extraction leads to reduction in levels needs to be explored.
- Pretty much all hot spots are affected positively to the minimum.



$d$
$k$
$|V|$



$d$
$k$
$|V|$



$d$
$k$
$|V|$

# Finding Clique of $k$ size in graph $G$ with $n < 2k$

- Sort vertices by degree in descending order.
- Place top $k$ vertices in set $S_1$.
- place rest of the vertices in set $S_2$.
- Let $C_1$ be sum of edges of vertices in set $S_1$.
- Let $C_2$ be the sum of edges of vertices in set $S_2$.
- Let $k'$ $(k' < k)$ be the number of vertices in set $S_2$.
- Let $E_1 = C_1$ - $(k(k-1))/2$.
- Let $E_2 = C_2$ - $(k'(k'-1))/2$.
- At least $E_1$ edges of $S_1$ vertices are connected to $S_2$ vertices.
- At least $E_2$ edges of $S_2$ vertices are connected to $S_1$ vertices.
- Actual number of edges that connects vertices from $S_1$ with $S_2$ are at least $E_1$.

- Actual number of edges that connects vertices from $S_1$ with $S_2$ are exactly $E_1$ when $S_1$ has clique of $k$.

- Actual number of edges that connects vertices from $S_2$ with $S_1$ are at least $E_2$.

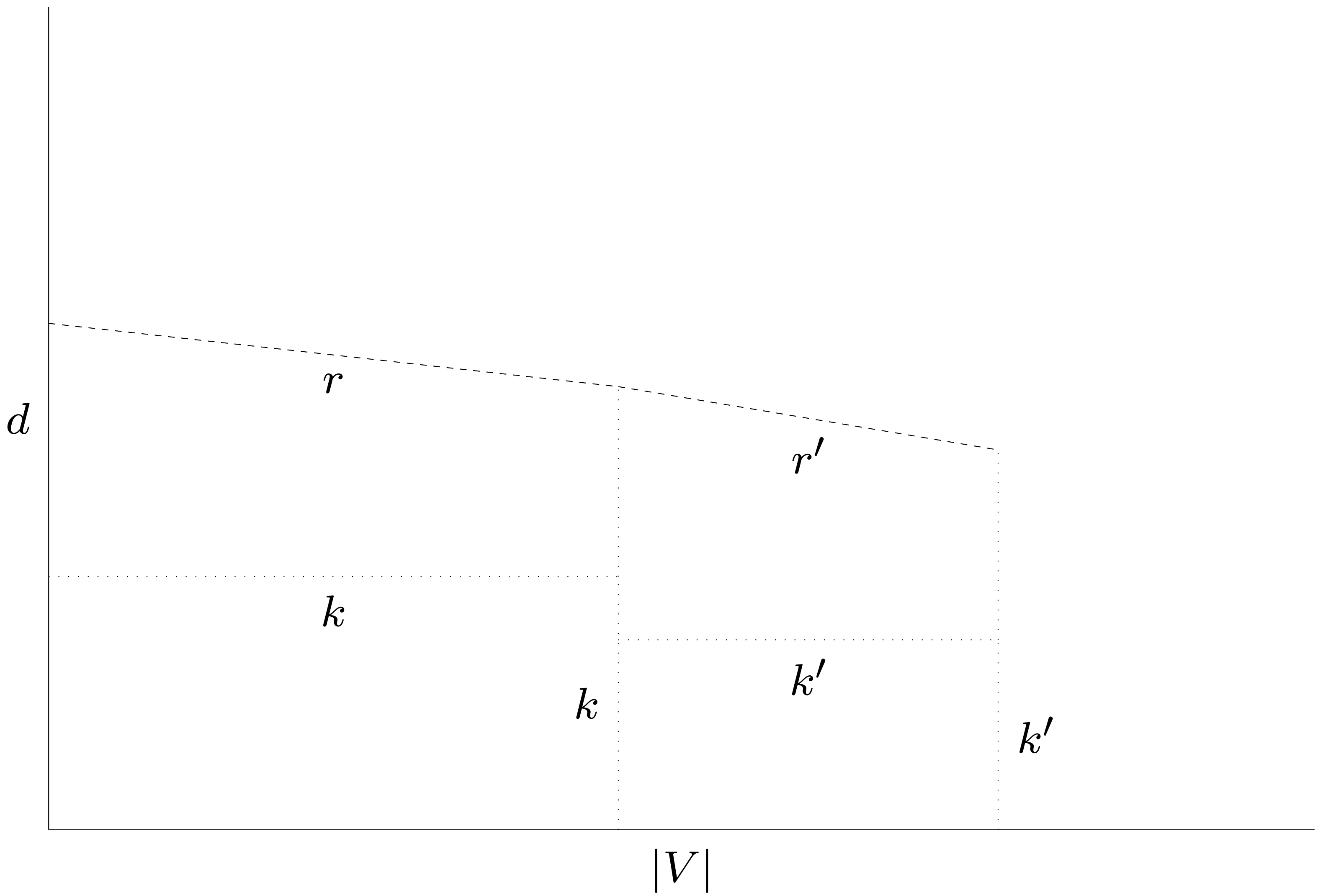
- Actual number of edges that connects vertices from $S_2$ with $S_1$ are exactly $E_2$ when $S_2$ has clique of $k'$.

- If $G$ has clique of $k$, then $E_1$ must be greater than or equal to $E_2$. Otherwise there exist no clique of size $k$.

- Satisfying this criteria does not mean that $k$ size clique exist since actual $E_2$ might be higher if $S_2$ does not form the clique of $k'$.

- Removing vertices with edges lower than $k$ narrows the gap between $E_2$ and actual $E_2$.

- Removing vertices with edges lower than $k$ could enable this check to be applied when graph $G$ has more than $2k$ vertices but there exists enough vertices with less than $k$ edges.


d
r
r′
k
k
k′
k′
|V|

**Algorithm 10** $FindMaximumCliqueSize$ : $\mathsf{O}(\mathsf{n}^{(\log(\mathsf{n}))})$

```
function FindMaximumCliqueSize(G(V, E), k)
    kMax ← max(k − 1, 0)
    partitions ← {}
    H ← G
    while (|H| > kMax) do
        while |(p := ExtractPartition(H))| > 0 do
            partitions ← partitions ∪ p
            H ← H − p
            kMax ← max(kMax − 1, 0)
        end while
        if (H = {}) then
            break
        end if
        if (|H| < 2 ∗ kMax) then
            k_1 ← |H| − kMax
            E_1 ← kMax + SumTopVertexEdges(H)
            E_2 ← k_1 + SumOtherVertexEdges(H)
            if ((E_1 − kMax^2) < (E_2 − k_1^2)) then
```

**break**
**end if**
**end if**
$v \leftarrow PickAVertexWithLeastDegree(H)$
$G' \leftarrow neighborhood(H, v, kMax)$
**if** $(|G'| >= kMax)$ **then**
$search \leftarrow true$
**for all** $v' \in \{G - H\}$ **do**
$G'' \leftarrow neighborhood(G, v')$
**if** $(isSubgraph(G'', G')$ **then**
$search \leftarrow false$
**end if**
**end for**
**if** $search$ **then**
$(found_1, kMax_1) \leftarrow FindMaximumCliqueSize(G', kMax)$
**if** $(found_1$ and $((kMax_1 + 1) > kMax))$ **then**
$kMax \leftarrow kMax_1 + 1$
**end if**
**end if**
**end if**

$G' \leftarrow neighborhood(H, v)$
$H \leftarrow \{H - v\}$
**for all** $v' \in H$ **do**
$G'' \leftarrow neighborhood(H, v')$
**if** $(isSubgraph(G', G'')$ **then**
$H \leftarrow \{H - v'\}$
**end if**
**end for**
**end while**
$kMax \leftarrow |partitions| + kMax$
**return** $(kMax >= k, kMax)$
**end function**

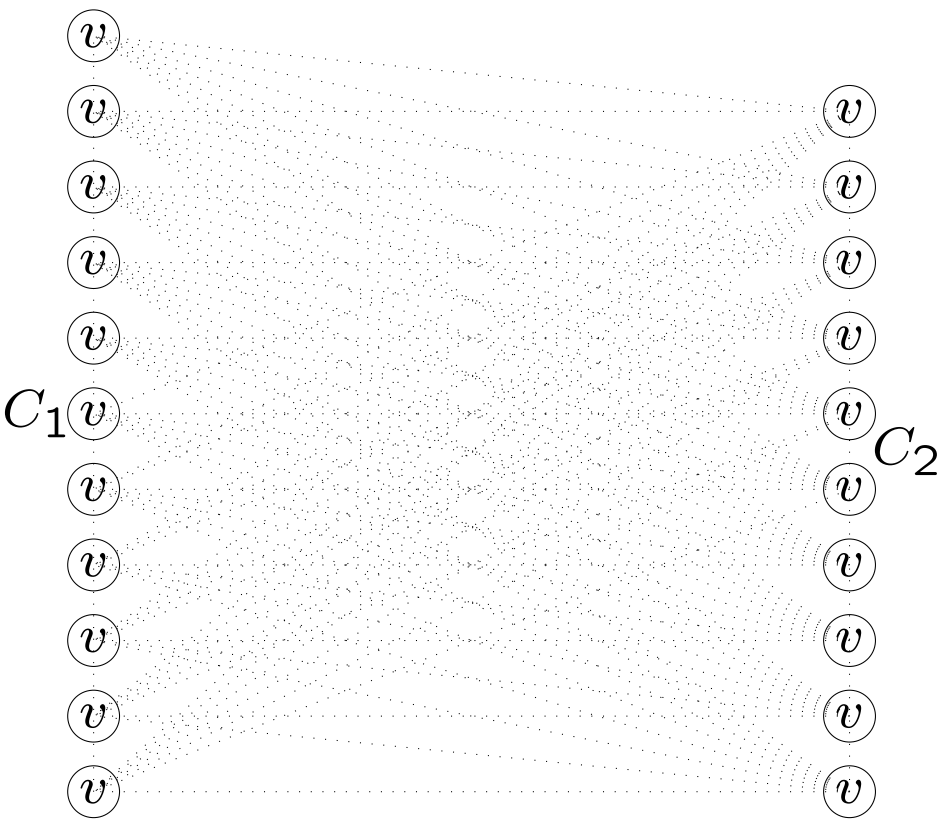
# Graphs that resist partition extraction

- Graph $G$ that contains more than one independent graphs as part of it.
- The resistance is only at the top level. Once a vertex is selected, the selected vertex's graph gets processed normally without any additional time complexity.
- Failed partition extraction at top level gets done at immediate next level unless partition-pivot vertex is not a neighbor of selected vertex. Eventually, this partition gets extracted at lower levels without altering time complexity.

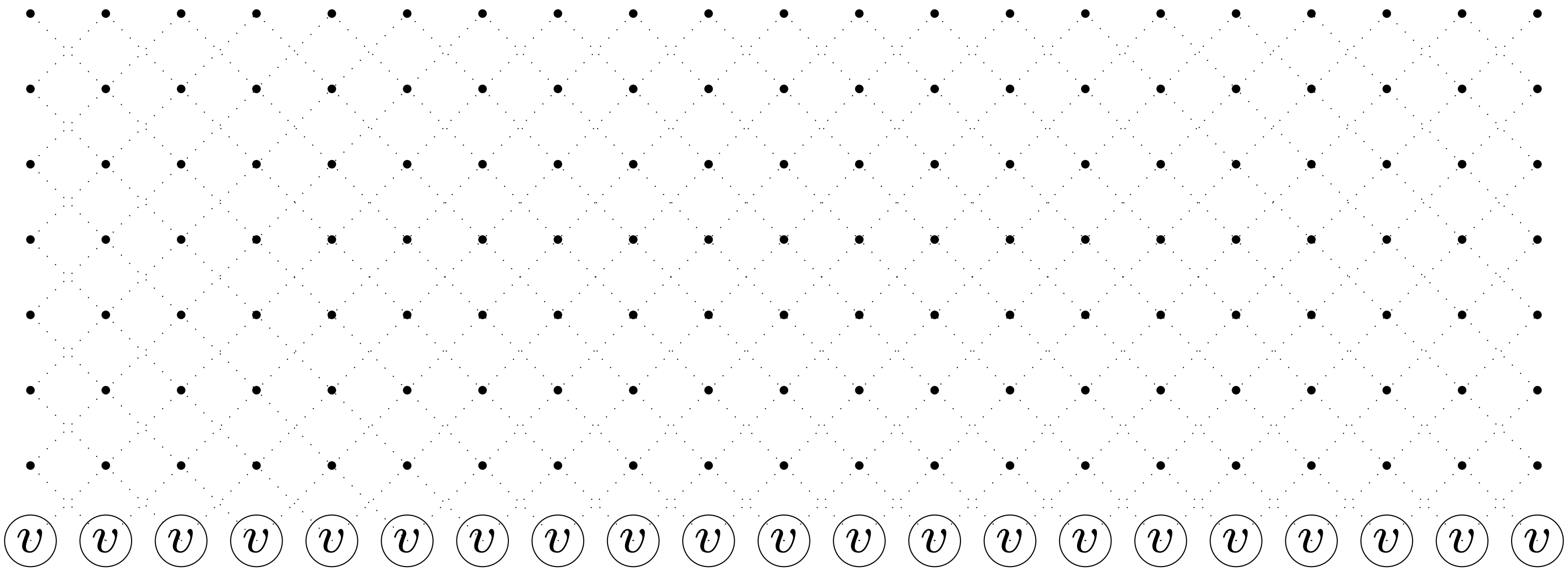
## Graphs that minimally resist partition extraction as well as bottom-up processing

- Graph $G$ that contains more than one clique sets.
  - Resistance goes away once small set of vertices are processed.
  - Their neighborhood graph collapses in polynomial time.
- Graph $G$ that exhibits shape of fence.
  - These fence graph's neighborhood graph exhibits steep slope and their size is approximately half of parent vertex edge count.
  - Neighborhood graph collapses in near polynomial time.

# Clique sets graph



# Fence graph



# Folding point

- This is a point where remaining active graph becomes neighborhood graph of one of the processed vertex.
- At this point further processing stops.
- Folding point could be within the last $K-1$ active vertices; processing stops at $K$ vertices since $K-1$ vertices can't produce clique of size $K$.

# Time complexity when $N < 2k$ having partition extraction applied ahead

- This case is considered for set of graphs such that $N < 2k$ where $N = |G|$ and no further partition can be extracted.

- Only graphs (non multipartite graph) that can resist partition extraction and satisfies $N < 2k$ would be here.

- Removing a vertex that does not have $k$ edges is of $O(n)$. So removing set of vertices that does not have $k$ edges is $O(n^2)$

- Any graph that does not meet edge count criteria will not have clique of size $k$ and the processing stops here. Time complexity is $O(n)$.

- Clique sets which resist partition extraction could also satisfy edge count criteria; but they collapse at neighborhood graph processing and after few vertices are processed. Time complexity is polynomial.

- If the graph is neither multipartite (friendly to partition extraction) nor clique sets and then they barely meets edge criteria. These graphs mostly have the shape of fence. These graphs collapse at neighborhood graph processing and after few vertices are processed. Time complexity is polynomial.

- If the graph is none of the above, then vertex with least degree does not yield a neighborhood graph that could have a clique of $k-1$. Time complexity is $O(n^3)$

- So, the time complexity for finding clique of $k$ where $N < 2K$ and partitions can not be extracted is polynomial.

# Counting & Enumerating all maximum cliques

- Partition extraction
  - Extracts complete partition
  - At least one vertex in the partition is connected to all vertices in all other partitions.
  - Partition may contain one or more vertex that is not connected to all other vertices in other partitions.
- When a active vertex's neighborhood graph is explored, a partition is created with the current active vertex. This partition should also contain vertices that have same neighborhood but not connected to active vertex that is being explored.
- When a multipartite graph with largest partition is found and there is no more vertex to be explored:
  - Extract proper complete partition. if the vertex that is connected to some of the vertices is not part of the clique then exclude it; otherwise extract a limited complete multipartite graph containing it.
  - There may be more than one complete multipartite graph.

  - Number of partitions is nothing but the clique size
  - Product of each partition size gives number of cliques in that multipartite graph for counting.
  - Multi-partite graph contains all the cliques that is explored.
- Enumerating all multipartite graphs with given number of partitions (maximum clique size) in $G$ gives
  - Total count of maximum cliques.
  - All maximum cliques (in compressed format : in the form of multipartite graph).

# Algorithm enhancements

- Find the maximum clique size.
- At least one maximum clique
- Count of all maximum cliques present
- Enumerate and list all maximum cliques in the form of set of complete multipartite graphs.

**Algorithm 11** $MaximumClique$ : $\mathsf{O}(\mathsf{n}^{(\log(\mathsf{n}))})$

**function** MaximumClique($G(V, E), k, partitions, op$)
    $kMax \leftarrow max(k - 1, 0)$
    $H \leftarrow G$
    **while** $(|H| > kMax)$ **do**
        **while** $|(p := ExtractPartition(H))| > 0$ **do**
            $partitions \leftarrow partitions \cup p$
            $H \leftarrow H - p$
            $kMax \leftarrow max(kMax - 1, 0)$
        **end while**

        **if** $(H = \{\})$ **then**
            **if** $(kMax = 0)$ **then**
                ▷ Count or Enumerate or etc...
            **end if**
            **break**
        **end if**

        **if** $(|H| < 2 * kMax)$ **then**

$k_1 \leftarrow |H| - kMax$<br>
$E_1 \leftarrow kMax + SumTopVertexEdges(H)$<br>
$E_2 \leftarrow k_1 + SumOtherVertexEdges(H)$<br>
**if** $((E_1 - kMax^2) < (E_2 - k_1^2))$ **then**<br>
    **break**<br>
**end if**<br>
**end if**<br>
<br>
$v \leftarrow PickAVertexWithLeastDegree(H)$<br>
$pp \leftarrow partitions \cup \{v, ...\}$<br>
$G' \leftarrow neighborhood(H, v, kMax)$<br>
**if** $(|G'| >= kMax)$ **then**<br>
    $search \leftarrow (op$ **not in** $\{Count, Enumerate\})$<br>
    **for all** $v' \in \{G - H\}$ **do**<br>
        $G'' \leftarrow neighborhood(G, v')$<br>
        **if** $(isSubgraph(G'', G')$ **then**<br>
            $search \leftarrow false$<br>
        **end if**<br>
    **end for**

**if** $(search)$ **then**
$(found_1, kMax_1) \leftarrow MaximumClique(G', kMax, pp, op)$
**if** $(found_1$ and $((kMax_1 + 1) > kMax))$ **then**
$kMax \leftarrow kMax_1 + 1$
**end if**
**end if**
**end if**
$G' \leftarrow neighborhood(H, v)$
$H \leftarrow \{H - v\}$
**for all** $v' \in H$ **do**
$G'' \leftarrow neighborhood(H, v')$
**if** $(isSubgraph(G', G''))$ **then**
$H \leftarrow \{H - v'\}$
**end if**
**end for**
**end while**
$kMax \leftarrow |partitions| + kMax$
**return** $(kMax >= k, kMax)$
**end function**

## Space complexity in terms of $|G|$

- A graph is created at each recursion level. Space requirement for graph is $O(n^2)$.
- At each level fixed set of array is created based on $|G|$. Space requirement is $O(n)$.
- Algorithm recursion depth is related to $|G|$. Though recursion depth is related to $|G|$, the actual one is substantially smaller portion of $|G|$.
- At each level, input size $n$ is smaller than previous level.
- Therefore the space complexity is lower than $O(n^3)$.

## Space optimization $|G|$

- Though the space complexity is lower than $O(n^3)$, it is still larger than a computer system can provide for when $N$ is in 1000's.
- When algorithm runs out of space (RAM), it could free-up top level intermediate graph(s) on need basis. When inner level returns, the freed up graph can be reconstructed from top most level's input graph.
- This would avoid flushing RAM contents in to disk storage which is costly in many order of magnitude.

Algorithm sensitivity to CPU data cache.

- Access speed of CPU data-cache is many order of magnitude faster than accessing system RAM.
- Modern computer(s) have large enough CPU data-cache to handle graph(s) with 100's of vertices without frequently swapping between CPU data-cache and system RAM.
- As the $|G|$ size increases, CPU data-cache size is not sufficient enough to hold the entire graph for few levels. At this instance, algorithm speed is entirely depends on the speed of accessing system RAM.
- Therefore, the wall clock time to process graphs with 1000's of vertices is many order slower even though the time complexity is same.

## Number of proper complete multipartite graphs in G(V,E) relation to $|V|$.

- Trivial complete multipartite graph
  - All partitions are trivial. (i.e.) Each partition contains exactly one vertex in it.
  - Number of cliques in a trivial complete multipartite graph is exactly one.
- Proper complete multipartite graph
  - A complete multipartite graph which is not a subgraph of another complete multipartite graph.
  - There is no other complete multipartite graph which exist can be combined to form a new complete multipartite graph.
- Trivial complete multipartite graph can also be a proper complete multipartite graph.
- Complexity
  - Fix clique size to K
  - Partition the set of vertices into two groups.

- Take all vertices of one group and create as many proper complete (k-1)-partite graph as possible.
- For each vertex in second group, create one trivial k-partite graph from each proper complete k-partite graphs as follows.
  * Create one trivial (k-1)-partite graph from each proper complete (k-1)-partite graph.
  * the same k-1 partite graph can't be used more than once.
  * Add the vertex from second group to form a trivial complete k-partite graph.
- Since, we can create number of vertices in the second group times number of proper (k-1)-partite graphs, and all are proper complete k-partite graph, complexity is $O(n^2)$
- Though the complexity of number of instances is $O(n^2)$, these instances can be packed in $O(n)$ packs.

## What is the maximum number of cliques a graph $G(V, E)$ can have and at what clique size?

- $N \bmod 3 \equiv 0$
  - $N/3$ partitions; each having 3 vertices in it.
  - Maximum clique size : $N/3$
  - Number of cliques with size $N/3$ is $(N/3)^3$
- $N \bmod 3 \equiv 1$
  - $((N-1)/3)+1$ partitions; $(N-4)/3$ partitions having 3 vertices and 2 partitions having two vertices.
  - Maximum clique size : $((N-1)/3)+1$
  - Number of cliques with size $((N-1)/3)+1$ is $((N-4)/3)^3 \times 2^2$
- $N \bmod 3 \equiv 2$
  - $((N-2)/3)+1$ partitions; $(N-2)/3$ partitions having 3 vertices and one partition having two vertices.
  - Maximum clique size : $((N-2)/3)+1$
  - Number of cliques with size $((N-2)/3)+1$ is $((N-2)/3)^3 \times 2$

## Time complexity for $G(V, E)$ with $N$ vertices

- For worst case analysis, we take maximum clique size $K$ is same as number of partition $P$. If the $K$ is lower than $P$ then it has lower time complexity compared to where $K = P$.
- Number of partitions ($P$) in the graph: this determines the maximum size of the clique. Also determines the depth the iterations.
- Partition size ($S$): Minimum partition size is 1; maximum possible partition size is $(N - P + 1)$
- Sum of vertices in all partitions equals $N$.
- Average partition size is $N/P$
- At each iteration, we only need to explore $(N_i - 2 * K_i + 1)$.
- At each depth or partition extraction, $K_i$ gets reduced by 1 while $N_i$ gets reduced by at least 3.The iteration size reduces acutely.
- Further exploring is needed until $N_i$ becomes lesser than $2K_i$. At this point, remaining processing becomes polynomial.

- As vertices are processed and removed from active consideration, the remaining active graph becomes smaller, and starts exposing multi-partite graphs. This enables both partition extraction and neighborhood graph duplicate removal optimization to take place.
- Both number of partitions $P$ and average partition size $S$ are controlled by $N$ as $P * S = N$.
- $K$ is directly proportional to $P$.
- $K$ is inversely proportional to $S$.
- Folding point's location at each depth; Pushing folding point to within $2K - 1$ at each depth makes the overall depth shallow due acute convergence.
- Larger $P$
  - As the $P$ increases, $K_i$ at each level also increases.
  - It is only $(N_i - 2 * K_i + 1)$ needs to processed at each level, therefore the number of vertices needs to be explored becomes smaller at each level.
  - Larger $K$ requires each vertex to have larger number of edges. This causes convergence towards neighborhood graph with $2K_i$ vertices acute.

  - Also gap between $N_i$ and $2K_i$ is smaller; so the convergence towards neighborhood graph with $2K_i$ vertices is acute.
  - Large $K$ means smaller $S$; smaller $S$ brings folding point sooner than $2K_i$. This reduces the number of vertices that needs to be explored at each level.

- Smaller $P$
  - As the $P$ decreases, the required depth of the search decreases.
  - With larger partition size $S$, subsequent neighborhood graph becomes smaller and smaller acutely.
  - If the graph $G$ is denser then both partition extraction and neighborhood graph elimination (bottom up processing) works aggressively.
  - If the graph is sparse then it has lower time complexity.

- $P \approx S$
  - Sufficient enough depth to provide maximum possible breath at each level.
  - Even at this parameters, the convergence to neighborhood graph of $2K_i$ is acute; does not require to explore to the depth of $K$.

- Simplistic order here is $O(N^{\sqrt{N}-C})$ where $C > 0$.
- Since number of vertices explored at each level is at least $S$ lower than then previous level, the above time complexity can be rewritten as $O(N^{k(\sqrt{N}-C)})$ where $C > 0$ and $0 < k < 1$. As $C$ increases $k$ tends towards 1 and vice versa.

Observed Time complexity for $G(V, E)$ with $N$ vertices

- $C$ in the above time complexity reaches more than half of $\sqrt{N}$.
- For a graph $G(V, E)$ with 512 vertices, the time complexity is coming around $O(N^6)$. That is roughly around $O((2^9)^6) = O(2^{54})$.

## Minor optimization

- Let $v^1$ & $v^2$ are vertices of graph $G$.
- Let $G^1$ be neighborhood graph of $v^1$.
- Let $G^2$ be neighborhood graph of $v^2$.
- Let $S^1$ is set of vertices found only in $G^1$.
- Let $S^2$ is set of vertices found only in $G^2$.
- Let $S'$ is set of vertices found in both $G^1$ & $G^2$.
- Consider that $G^1$ is already explored.
- During the processing of $G^2$, we can discard any graph $G''$ which has all vertices from $S'$. This is because, $G''$ is already explored as part of $G^1$.

Execution statistics for keller5.clq
http://mat.gsia.cmu.edu/COLOR02/INSTANCES/keller5.clq

| | |
|---|---|
| Vertices | 776 |
| Edges | 225990 |
| Clique | 27 |
| Ticks(ms) | 2542521999 |
| Calls | 703285173499 |
| TwoNHits | 633190806735 |
| SubgraphHits | 1179362341357 |
| BtmUpHits | 495495356392 |
| BtmUpHits2 | 1317047970 |

| Depth | Calls |
|---|---|
| 1 | 1 |
| 2 | 401 |
| 3 | 142948 |
| 4 | 18402772 |
| 5 | 1047840586 |
| 6 | 26427005000 |
| 7 | 249569092300 |
| 8 | 400189686052 |
| 9 | 26026406477 |
| 10 | 6596962 |