Worst-Case Subgraph Connectivity

Ran Duan

My Research

- Basic Graph Optimization Problems
  - Connectivity
  - Shortest Path
  - Matching
  - Maximum Flow

- Exact Algorithms
- Approximate Algorithms
- Dynamic data structures

Basic Concepts and Notations

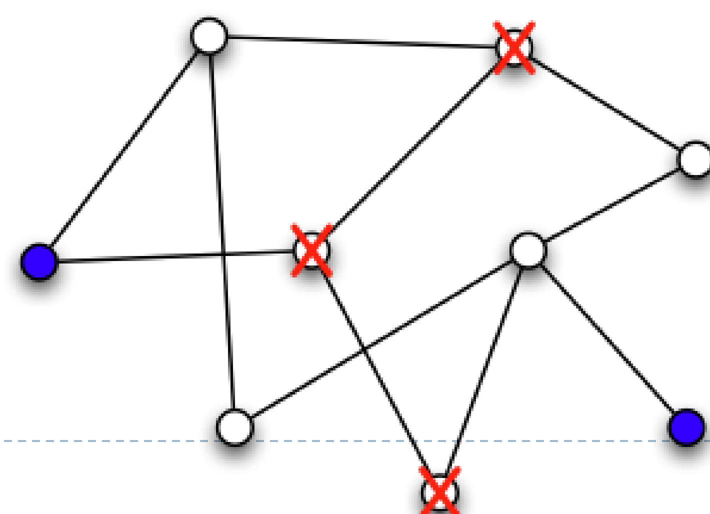
- G=(V, E): Primary graph we consider
  - n=|V|, m=|E|
- Weighted graph: $w : E \rightarrow \Re$
- Connectivity: whether there is a path between two vertices u,v (in undirected graphs).
- Shortest path: the path p connecting u and v minimizing $\sum_{e \in p} w(e)$

Traditional dynamic graph

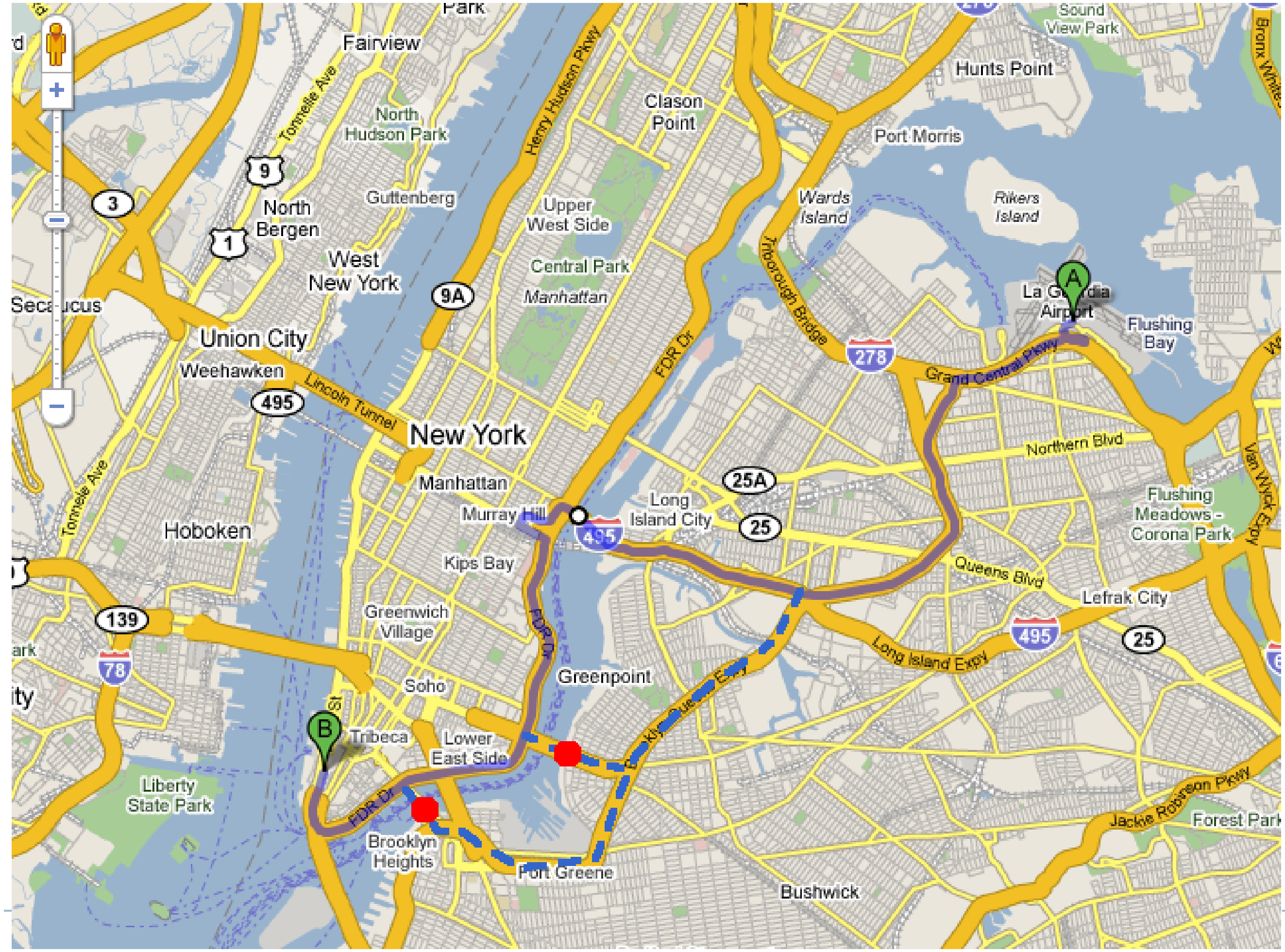
- Fully dynamic: we can insert and delete edges/vertices arbitrarily
- Decremental: only deletions
- Incremental: only insertions

Dynamic Subgraph Model

- There is a fixed underlying graph G, every vertex in G is in one of the two states "on" and "off".
- Construct a dynamic data structure:
  - Update: Switch a vertex "on" or "off".
  - Query: For a pair (u,v), answer connectivity/shortest path between u and v in the subgraph of G induced by the "on" vertices.



Motivation of dynamic subgraph model



Overview (Dynamic Subgraph Model)

- d-failure connectivity (STOC 2010)
  - The first d-vertex failure connectivity structure of query time only polynomial of d and log n.
  - Processing time when given d failed vertices: $\tilde{O}(d^{2c+4})$
  - Query time: O(d); Space: $\tilde{O}(mn^{1/c})$
  - c is an integer at least 1 and controls the time/space tradeoff
- Worst-case fully subgraph connectivity (ICALP 2010)
  - Subgraph connectivity structure of $\tilde{O}(m^{4/5})$ worst-case update time, with query time $\tilde{O}(m^{1/5})$
- Dual-failure shortest path (SODA 2009)
  - Two vertex failure shortest path structure of $O(n^2 \log^3 n)$ space and O(log n) query time.
- $\tilde{O}(\bullet)$ hides poly-logarithmic factors
  - For example, $\tilde{O}(n^2)$ means $O(n^2 \bullet \log^k n)$ for some constant k.

Overview of Shortest Path Results

- All-pair shortest path
  - Dijkstra's algorithm: $O(mn+n^2\log n)$
  - Pettie improves to $O(mn+n^2\log\log n)$
  - Floyd-Warshall algorithm: $O(n^3)$
  - Chan's: $O(n^3 \bullet \log^3\log n/\log^2 n)$
  - No real sub-cubic algorithm now

- Dynamic all-pair shortest path (edge update)
  - Demetrescu and Italiano/Thorup: update time $O(n^2\log^3 n)$

Overview of Dynamic Connectivity Results

- Edge update—amortized time
  - Holm, Lichtenberg, and Thorup: $O(\log^2 n)$
- Edge update—worst-case
  - Frederickson, Eppstein et al: $O(n^{1/2})$
  - (Kapron, King, Mountjoy: $O(\log^5 n)$, "randomized")
- Not suitable for vertex-update structure
- We can get faster update time for the subgraph model.

Two dynamic subgraph models

- d-failure model:
  - The number of "off" vertices is bounded by an integer d
  - It can be seen as a static structure, in which the query (u,v) is given with a set D of "off" vertices and $|D| \leq d$
- Real dynamic subgraph model:
  - We can change the status of any vertex in any time

Our Results

- d-failure model:
  - The first d-vertex failure connectivity structure of query time only polynomial of d and log n.
  - Two vertex failure shortest path structure of $O(n^2\log^3 n)$ space and $O(\log n)$ query time.
- Real dynamic subgraph model:
  - Subgraph connectivity structure of $\tilde{O}(m^{4/5})$ worst-case update time, with query time $\tilde{O}(m^{1/5})$

Our Results

- The first d-vertex failure connectivity structure of query time only polynomial of d and log n.
  - (Here $|D| \le d$, $n=|V|$, $m=|E|$.)
  - (c is an integer at least 1)

	Processing Time when given D	**Query Time**	**Size**
Our Structure	$\tilde{O}(d^{2c+4})$	$O(d)$	$\tilde{O}(m \cdot n^{1/c})$

Our Results

- The first d-vertex failure connectivity structure of query time only polynomial of d and log n.
  - (Here $|D| \le d$, $n=|V|$, $m=|E|$.)
  - (c is an integer at least 1)

		Processing Time when given D	Query Time	Size
Our Structure		$\tilde{O}(d^{2c+4})$	$O(d)$	$\tilde{O}(m \cdot n^{1/c})$
Trivial	Recompute	--	$O(m)$	$O(m)$
	Table	--	$O(1)$	$O(n^{d+2})$
Edge-failure structure [Pătraşcu and Thorup '2007]		$\tilde{O}(d \cdot n)$	$O(\log\log n)$	$O(m)$
Worst-case subgraph connecivity [Duan '2010]		$\tilde{O}(d \cdot m^{4/5})$	$\tilde{O}(m^{1/5})$	$\tilde{O}(m)$
Two-vertex failure distance structure [Duan & Pettie '2009]		--	$O(\log n)$	$\tilde{O}(n^d)$

New **Edge Failure** Structure

- As a component of the main structure, given a spanning tree, this structure can answer the connectivity when d edges fail.

	Processing Time	Query Time	Size	Construction Time
New edge failure structure	$O(d^2 \bullet \log\log n)$	$O(\log\log n)$	$\tilde{O}(m)$	$\tilde{O}(m)$
Edge-failure structure [Pătraşcu and Thorup '2007]	$\tilde{O}(d \bullet \log^2 n)$	$O(\log\log n)$	$O(m)$	Exponential
	$\tilde{O}(d \bullet \log^{2.5} n)$	$O(\log\log n)$	$O(m)$	Polynomial

- Our structure do not need to compute the sparsest cut, thus the construction is straight forward.

Our Results

- d-failure model:
  - The first d-vertex failure connectivity structure of query time only polynomial of d and log n.
  - Two vertex failure shortest path structure of $O(n^2 \log^3 n)$ space and $O(\log n)$ query time.
- Real dynamic subgraph model:
  - Subgraph connectivity structure of $\tilde{O}(m^{4/5})$ worst-case update time, with query time $\tilde{O}(m^{1/5})$

Dynamic Connectivity

	Edge Updates			Vertex Updates (Subgraph Model)		
	Update time	Query time	Space	Update time	Query time	Space
Amortized	$O(\log^2 n)$	$O(\log n/\log\log n)$	$O(m)$	$\tilde{O}(m^{2/3})$	$\tilde{O}(m^{1/3})$	$\tilde{O}(m^{4/3})$
	(Holm, Lichtenberg & Thorup 1998) (Thorup 2000)			(Chan, Pâtraşcu & Roditty 2008)		
Worst-Case	$O(n^{1/2})$	$O(1)$	$O(m)$	$\tilde{O}(m^{4/5})$	$\tilde{O}(m^{1/5})$	$\tilde{O}(m)$
	(Frederickson 1985 Eppstein et al 1992)			(Duan 2010)		

- Amortized time
  - Average running time per update in dynamic structures.

Dynamic Connectivity

	Edge Updates			Vertex Updates (Subgraph Model)		
	Update time	Query time	Space	Update time	Query time	Space
Amortized	$O(\log^2 n)$	$O(\log n/\log\log n)$	$O(m)$	$\tilde{O}(m^{2/3})$	$\tilde{O}(m^{1/3})$	$\tilde{O}(m^{4/3})$
				$\tilde{O}(m^{2/3})$	$\tilde{O}(m^{1/3})$	$O(m)$
	(Holm, Lichtenberg & Thorup 1998) (Thorup 2000)			(Chan, Pâtraşcu & Roditty 2008) (Duan 2010)		
Worst-Case	$O(n^{1/2})$	$O(1)$	$O(m)$	$\tilde{O}(m^{4/5})$	$\tilde{O}(m^{1/5})$	$\tilde{O}(m)$
	(Frederickson 1985 Eppstein et al 1992)			(Duan 2010)		

Algorithms Overview

- d-failure model:
  - d-failure connectivity
- Real dynamic subgraph model:
  - Worst-case connectivity

Difficulties and New Ideas

- Difficulty: we can't even spend O(1) time for every failed edge.

Difficulties and New Ideas

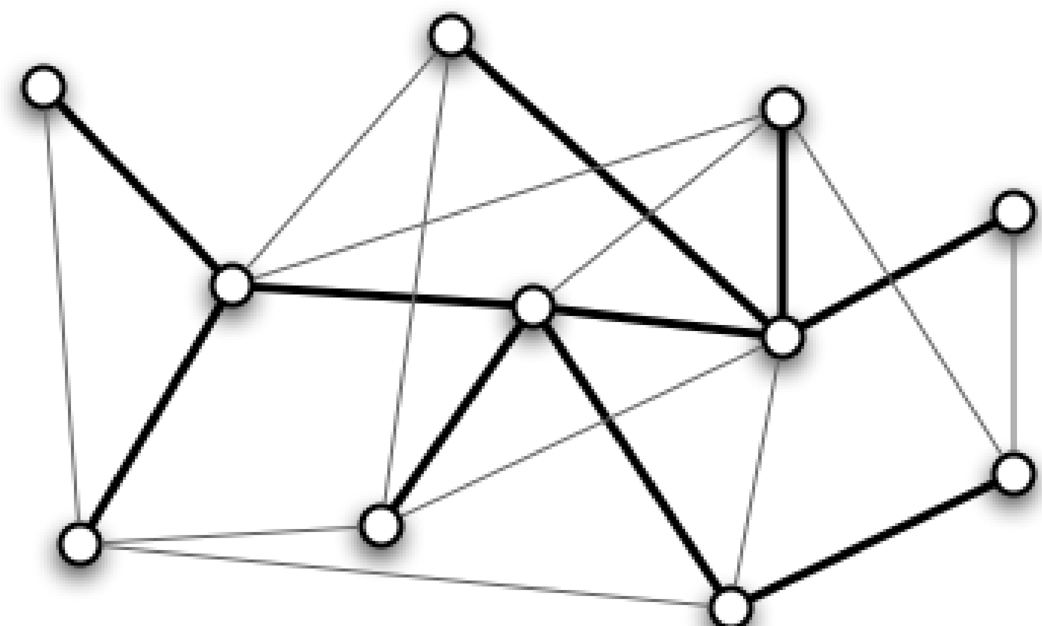
- Difficulty: we can't even spend O(1) time for every failed edge.
- A data structure where the deletion time is polynomial in degree of nodes **in a tree T**.
  - Non-tree edges are deleted implicitly.
  - If we have a degree-bound spanning tree T of G (degree smaller than s: $deg_T(v) \leq s$), we are already done.

Difficulties and New Ideas

- Difficulty: we can't even spend $O(1)$ time for every failed edge.
- A data structure where the deletion time is polynomial in degree of nodes **in a tree T**.
  - Non-tree edges are deleted implicitly.
  - If we have a degree-bound spanning tree T of G (degree smaller than s: $deg_T(v) \le s$), we are already done.
- A hierarchy of spanning forests such that the failed vertices are low-degree ($\le s$) in a set of trees for any D.
  - The degree threshold $s=d^{c+1}$ controls the time-space tradeoff.
  - The size of the hierarchy: $O(n^{1/c})$.
  - Time to delete failed vertices: $\tilde{O}(d^2 s^2)=\tilde{O}(d^{2c+4})$.

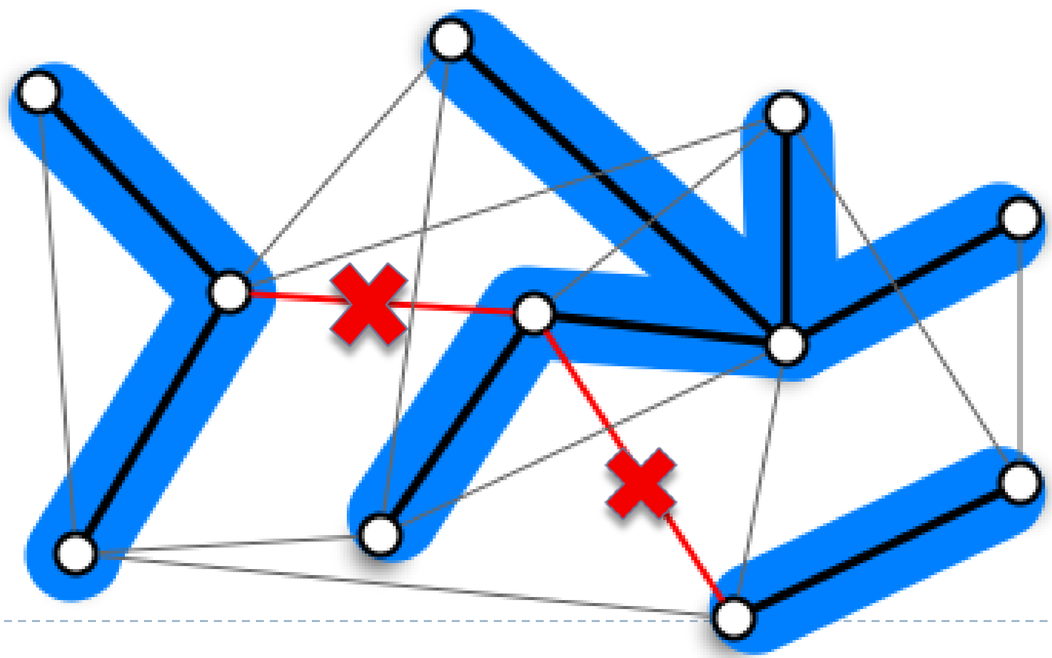
Basic Ideas

- Let T be a spanning tree of G.
  - Thick line– tree edges.
  - Thin line– non-tree edges.



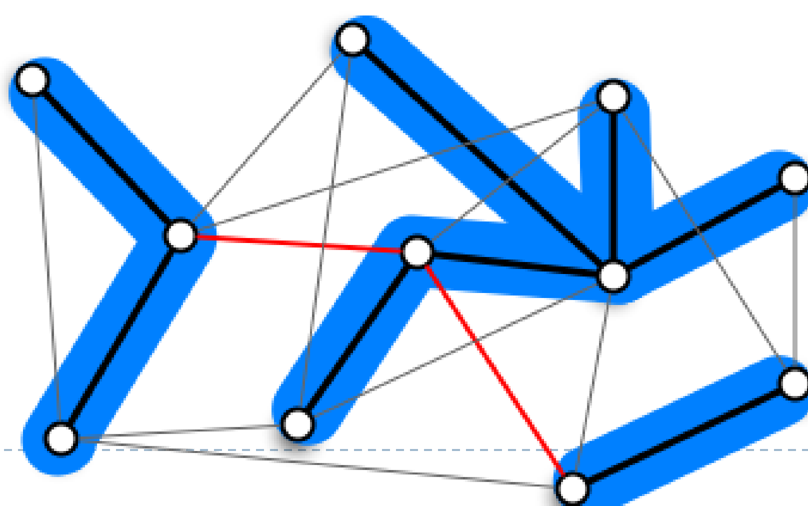
Basic Ideas

- Let T be a spanning tree of G. If we delete d' edges in T, T will be divided into d'+1 subtrees. We need to reconnect these subtrees.



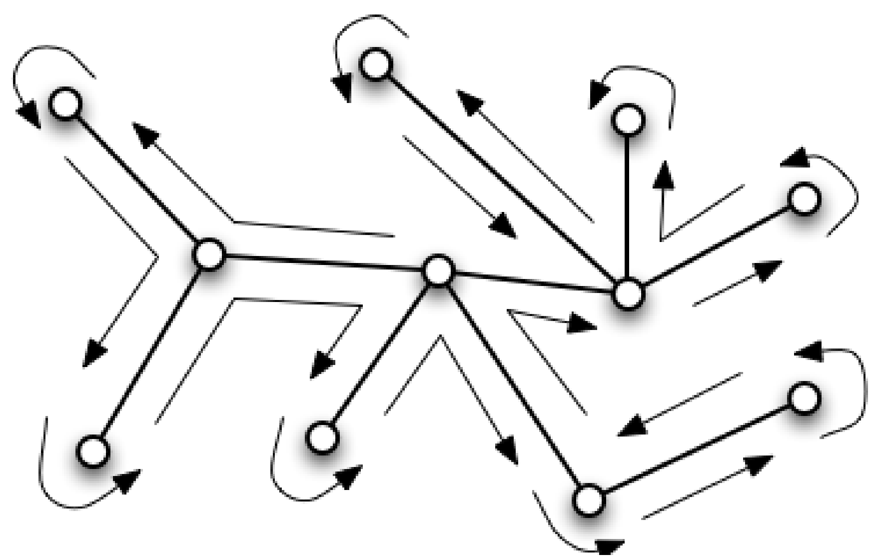
Basic Ideas

- We show that it takes O(loglog n) time to check whether two subtrees are connected by an edge,
  - So it takes $\tilde{O}(d'^2)$ time to check whether any pair of these d'+1 subtrees are connected by an edge.
  - Note that we cannot use the edge-failure structure by Pătraşcu and Thorup, since here we only consider the deletion of edges **in T** associated with the failed vertices, not the edges in G.



Reconnecting Subtrees

- Euler Tour of T:



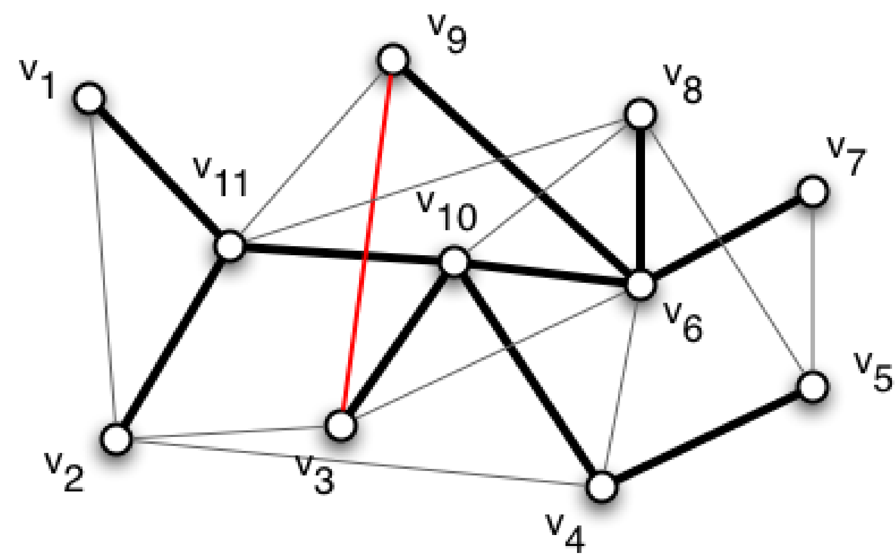
- Every vertex can appear many times in the Euler Tour, but we only keep any one of them for each vertex to form a ET-list:

  $v_1, \quad v_2, \quad \ldots \; v_n$

ET-list table:

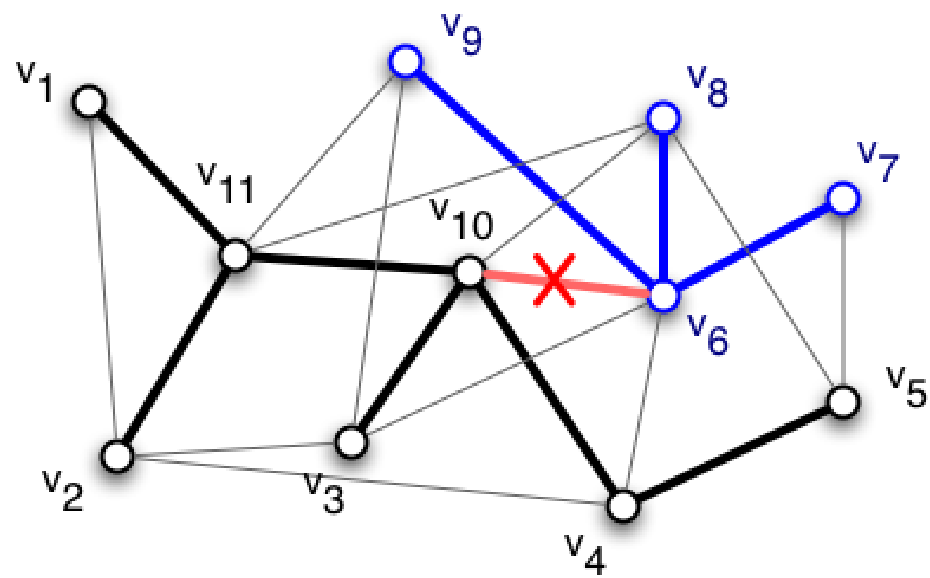
If there is a non-tree edge (v_i,v_j) in G, add element (i,j) into this table.

	1	2	3	4	5	6	7	8	9	10	11
1		X									
2	X		X	X							
3		X				X			X		
4		X				X					
5							X	X			
6			X	X							
7					X						
8					X					X	X
9			X								X
10								X			
11								X	X		



When we delete a tree edge, the ET-list will be divided into ≤3 parts.

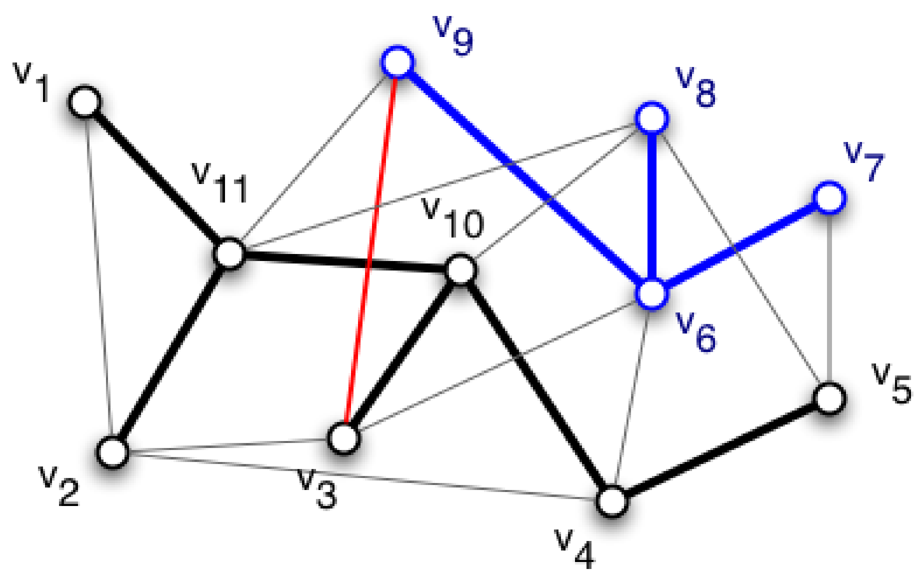
	1	2	3	4	5	6	7	8	9	10	11
1		X									
2	X		X	X							
3		X				X			X		
4		X				X					
5							X	X			
6			X	X							
7					X						
8					X					X	X
9			X								X
10								X			
11								X	X		



v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}

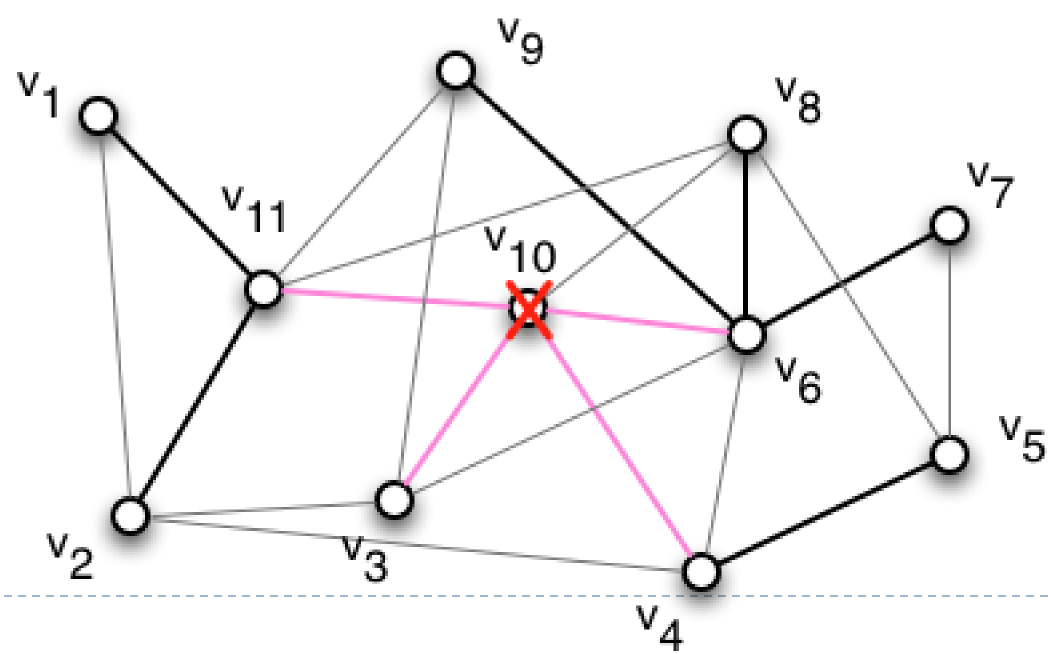
- It is a 2D range query to find edges to reconnect subtrees
- It takes O(loglog n) time to find an edge in every rectangle.
- So the time needed to reconnect after d tree-edge failures is $O(d^2 \log\log n)$.

	1	2	3	4	5	6	7	8	9	10	11
1		X									
2	X		X	X							
3		X				X			X		
4		X				X					
5							X	X			
6			X	X							
7					X						
8					X					X	X
9			X								X
10								X			
11								X	X		



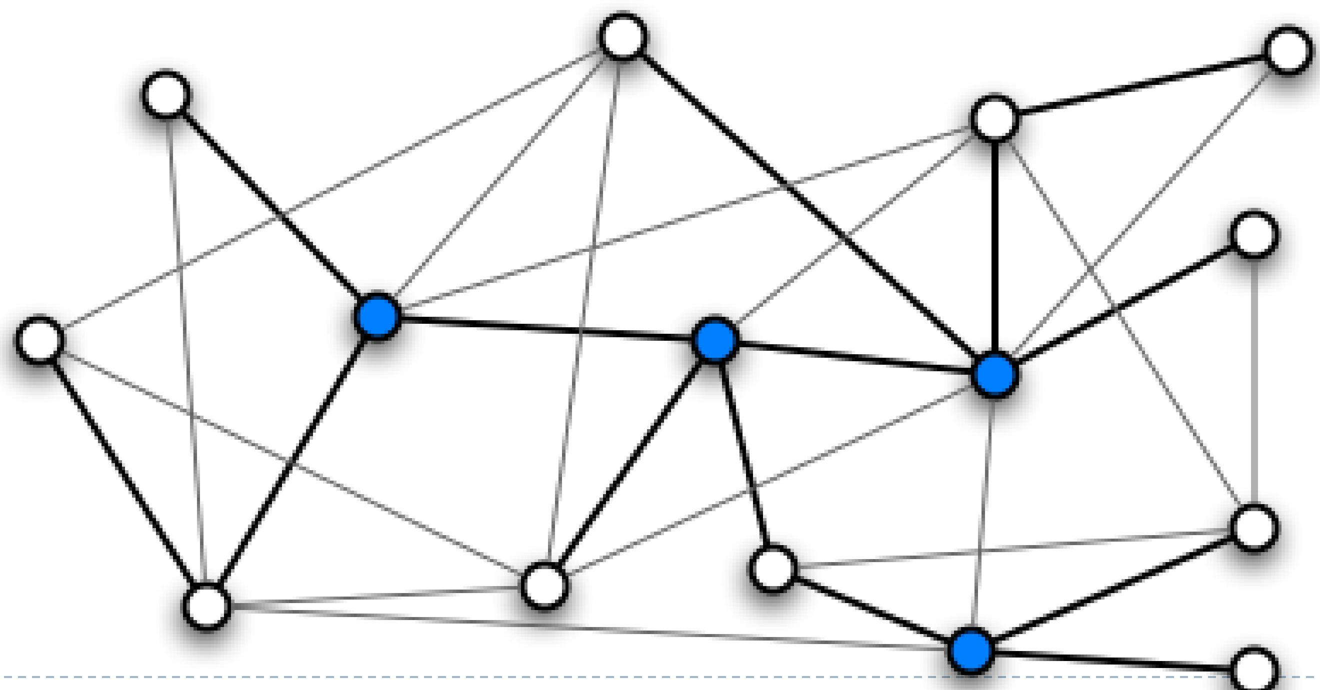
High-degree Vertices

- We choose an integer s such that $s>d^2$, $s=poly(d)$.
- If the degrees of all vertices are bounded by s in T, the time needed to reconnect the valid subtrees after d failures is $\tilde{O}(d^2s^2)$=poly(d), already done!

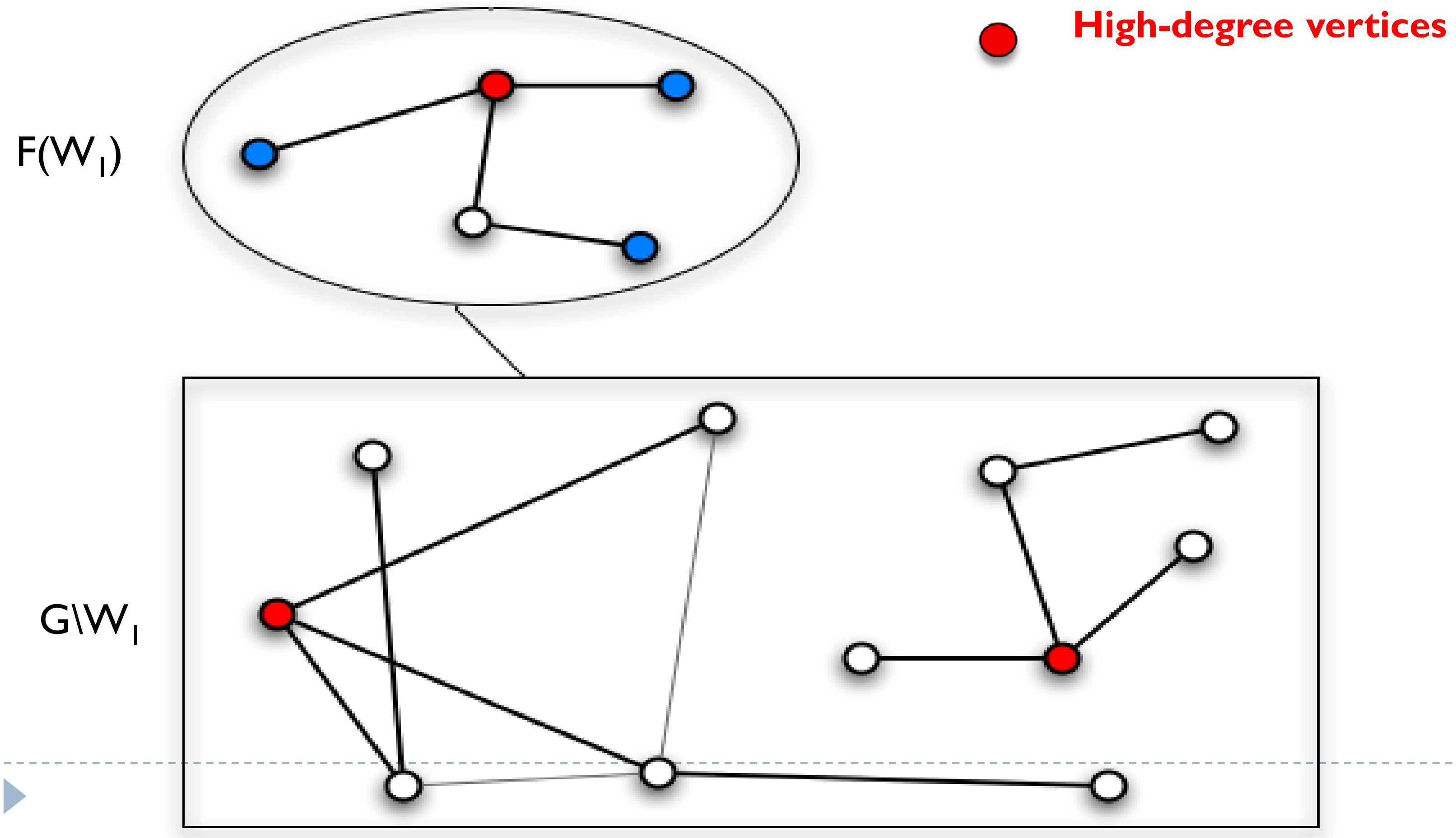


- We need to deal with the high-degree vertices **in T**.
  - High-degree vertices: degree larger than s in T,
  - Low-degree vertices: degree at most s in T.
- Since the number of edges in T is n-1, the number of high-degree vertices is at most 2n/s. (rough bound)

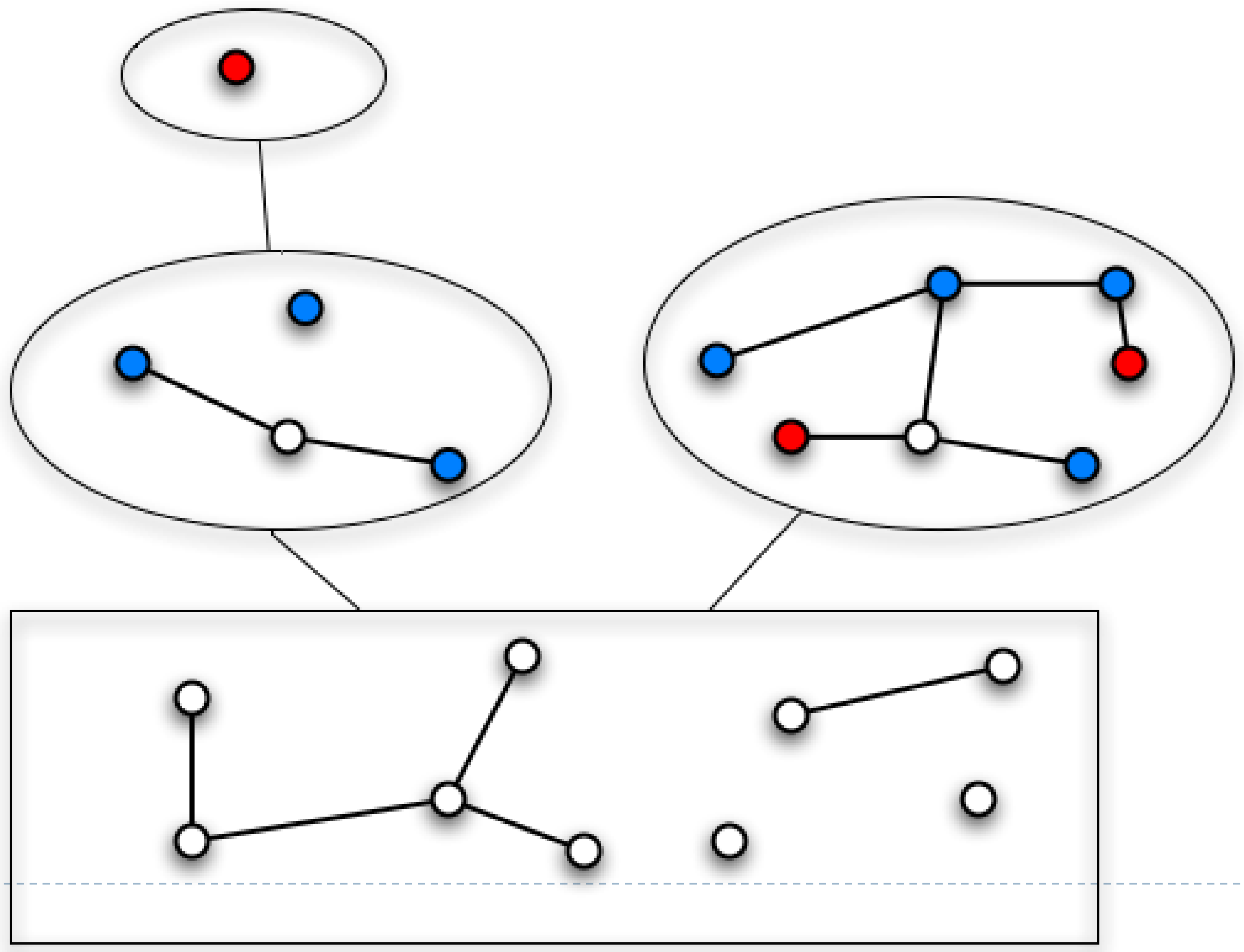
s=3 here



- We move these high-degree vertices to a higher level.
- Then reconnect the remaining vertices, which will create new high-degree vertices.
- Connecting high-level set will also create high-degree vertices.

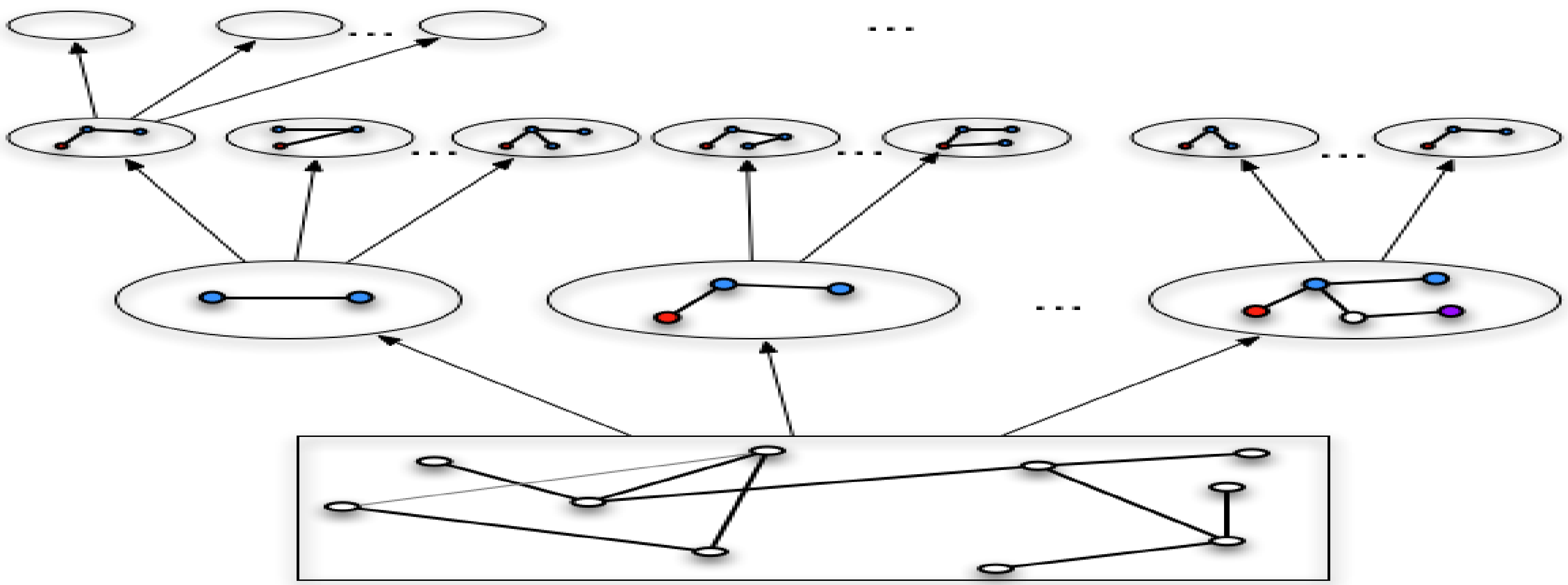


- Then move the new high-degree vertices to join with the previous high-degree vertices to form another set.

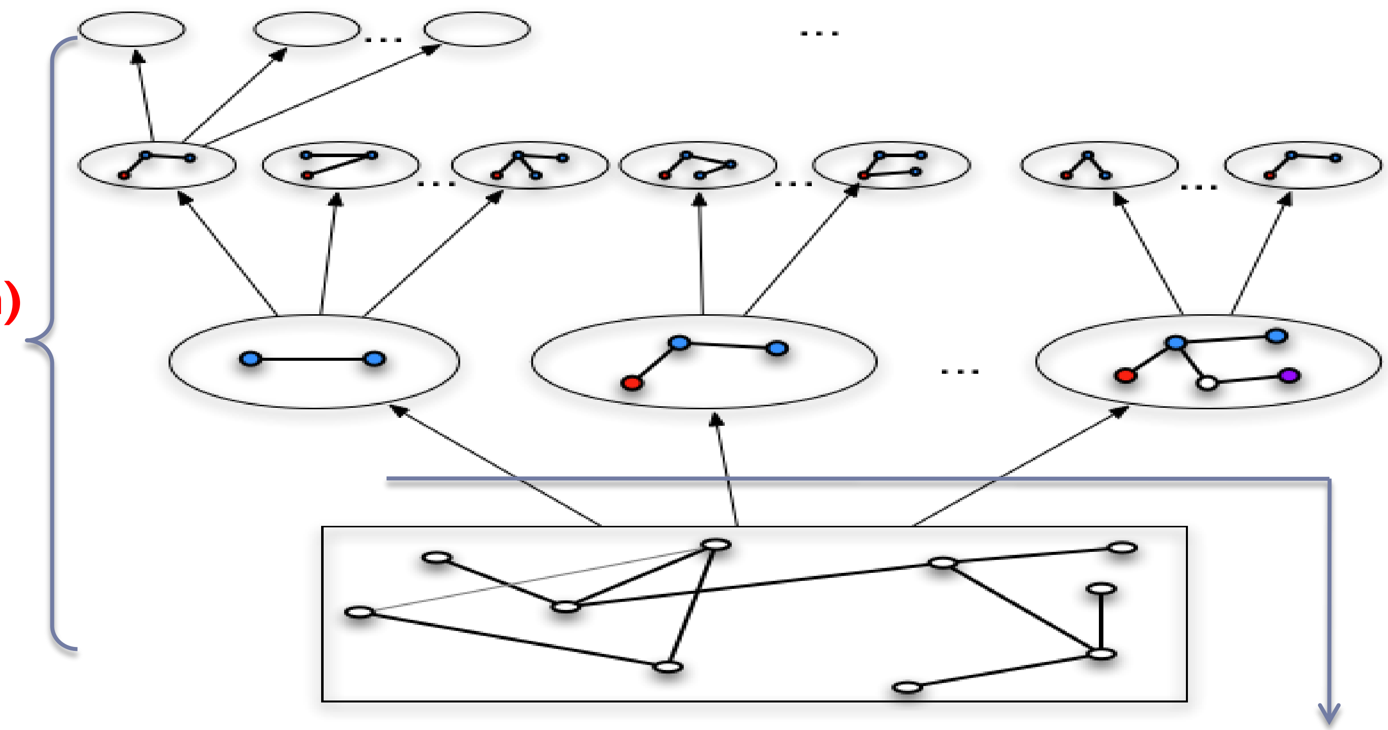


Construct the Hierarchy

- Move the high-degree vertices to higher level.
- Reconnect the remaining graph, if it still has high-degree vertices, also move them to higher level. Since there are at most d failures, this will repeat d times.
- Recursively deal with every high-level set.



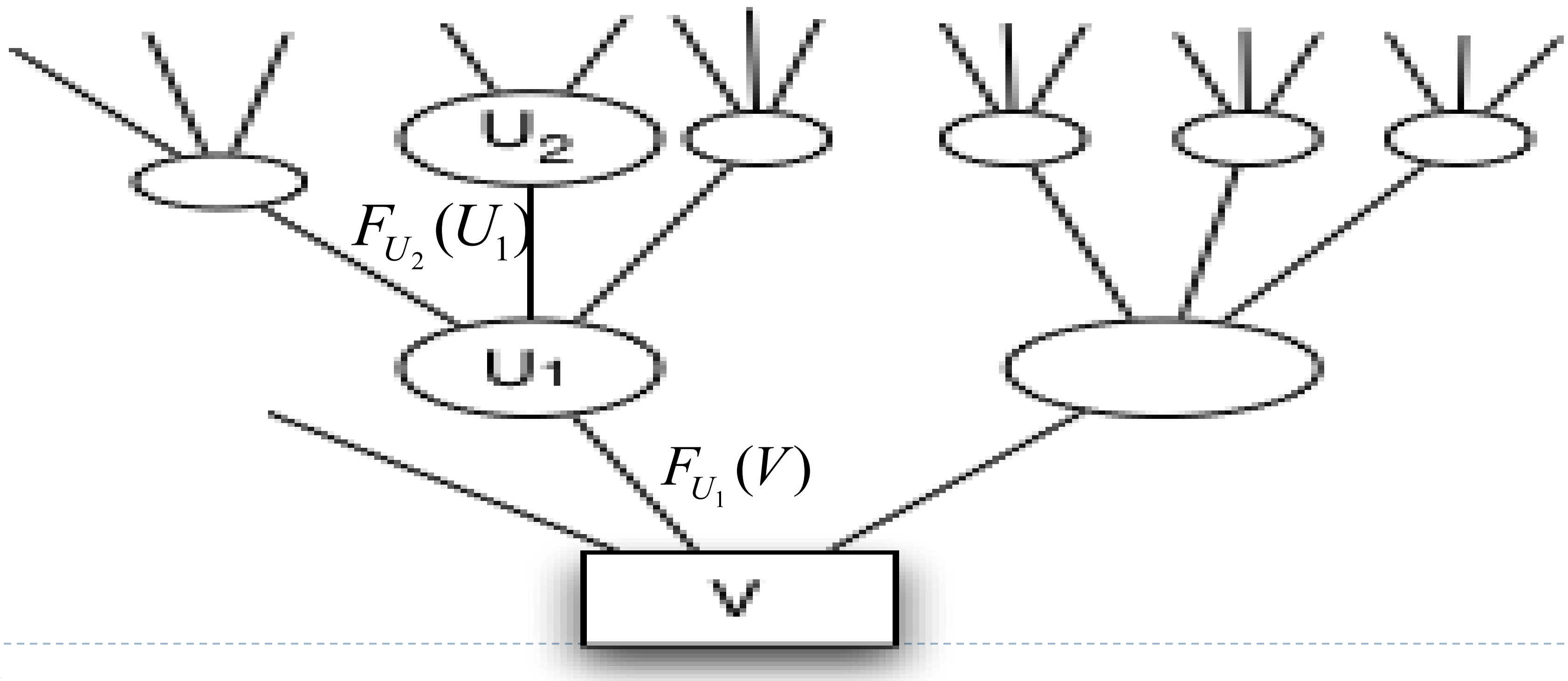
- d^{c+1}=s=high-degree threshold
- Number of hierarchy nodes: $d^{\log_{s/d} n} = O(n^{1/c})$
- So the parameter s controls time-space tradeoff



$O(\log_{s/d} n)$ levels

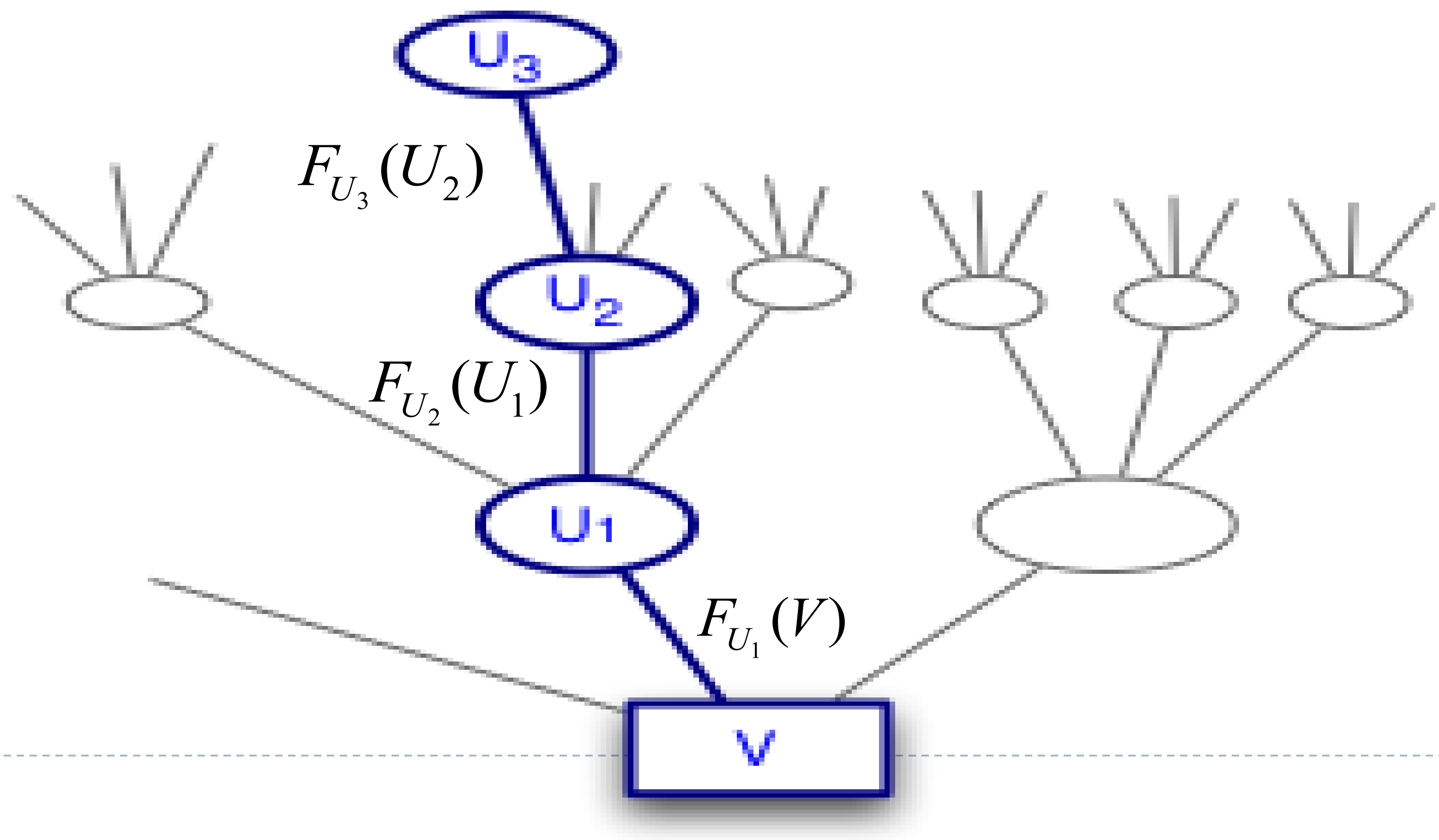
d children for every set is enough

- Nodes in the hierarchy identified with vertex sets.
- Define the forest on edge (U_i,U_{i+1}): $F_{U_{i+1}}(U_i)$ = the forest connecting $U_i \backslash U_{i+1}$ in the subgraph $G \backslash U_{i+1}$.



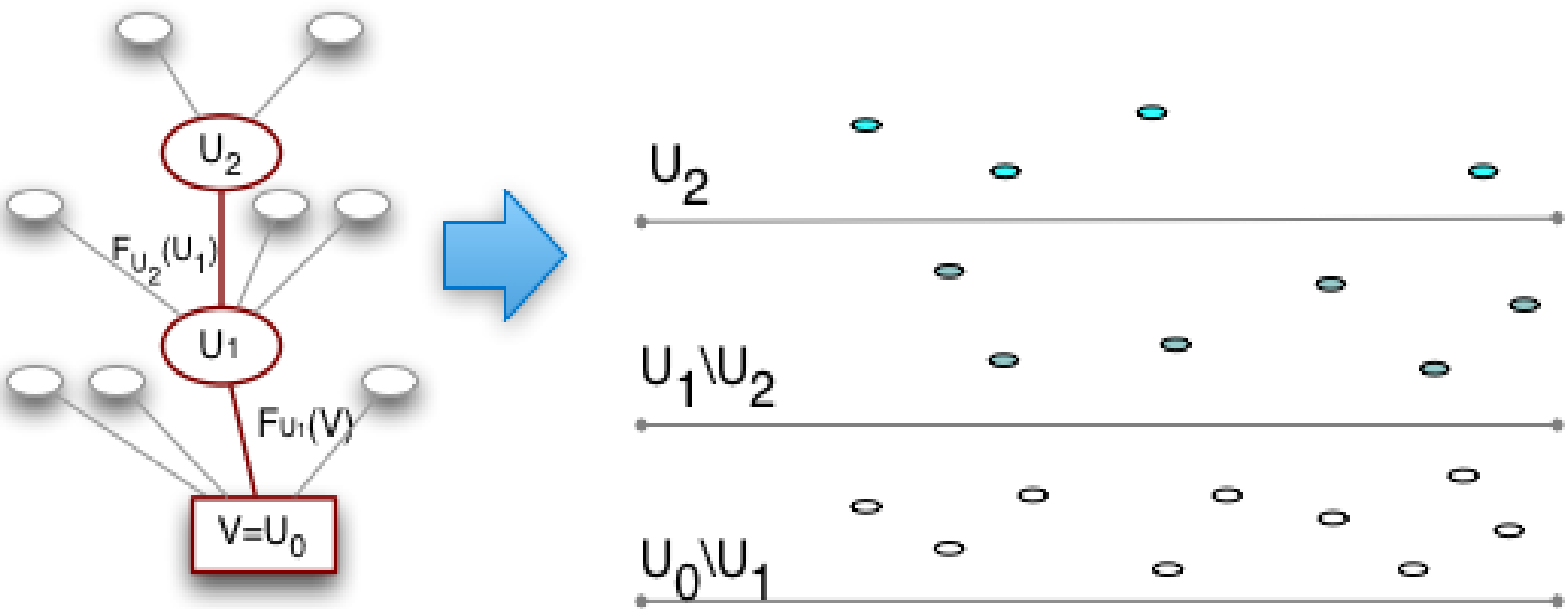
Key Property of the Hierarchy

- For all sets D of d failed vertices, there is a path in the hierarchy:
- V, U_1, U_2, …
- Such that all failed vertices are low-degree in $F_{U1}(V)$, $F_{U2}(U_1)$, …
- ($F_W(U)$= the forest connecting U\W in the subgraph G\W.)

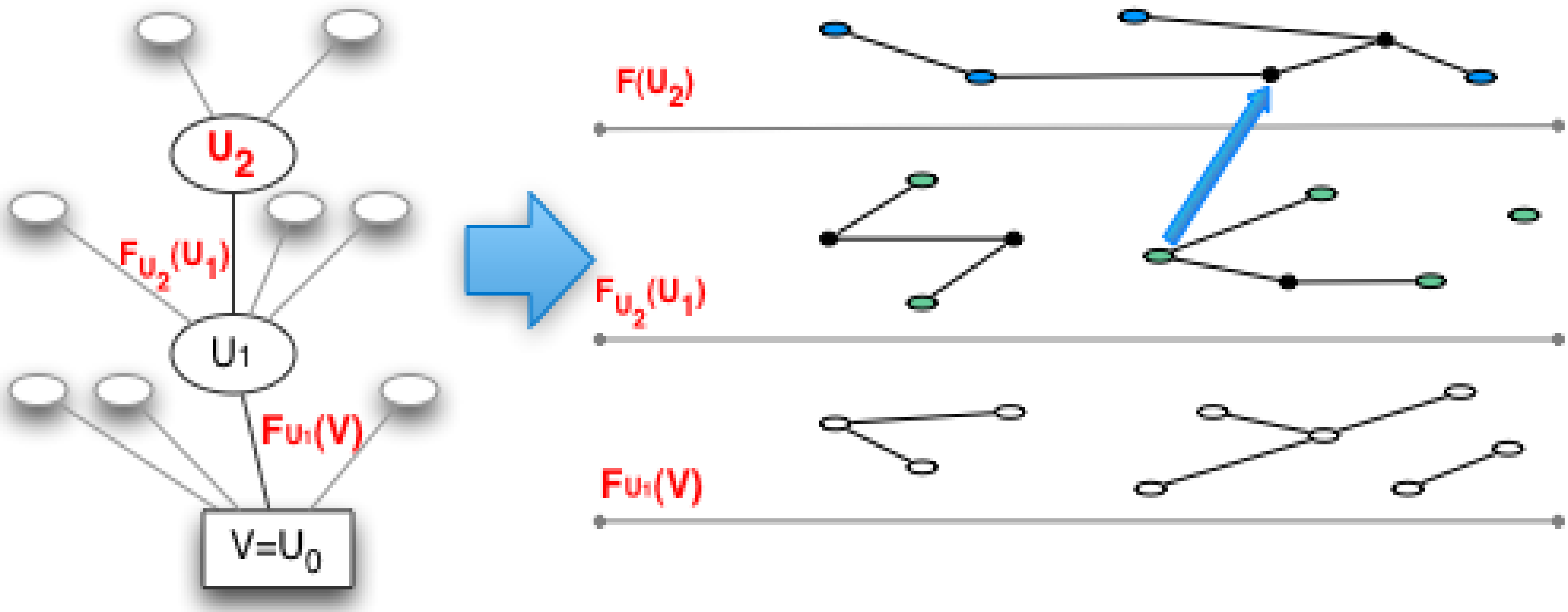


Inside the hierarchy

- For all paths in the hierarchy tree from the root to every node: $U_0(=V), U_1, \ldots, U_p$, where U_p is not necessarily a leaf.

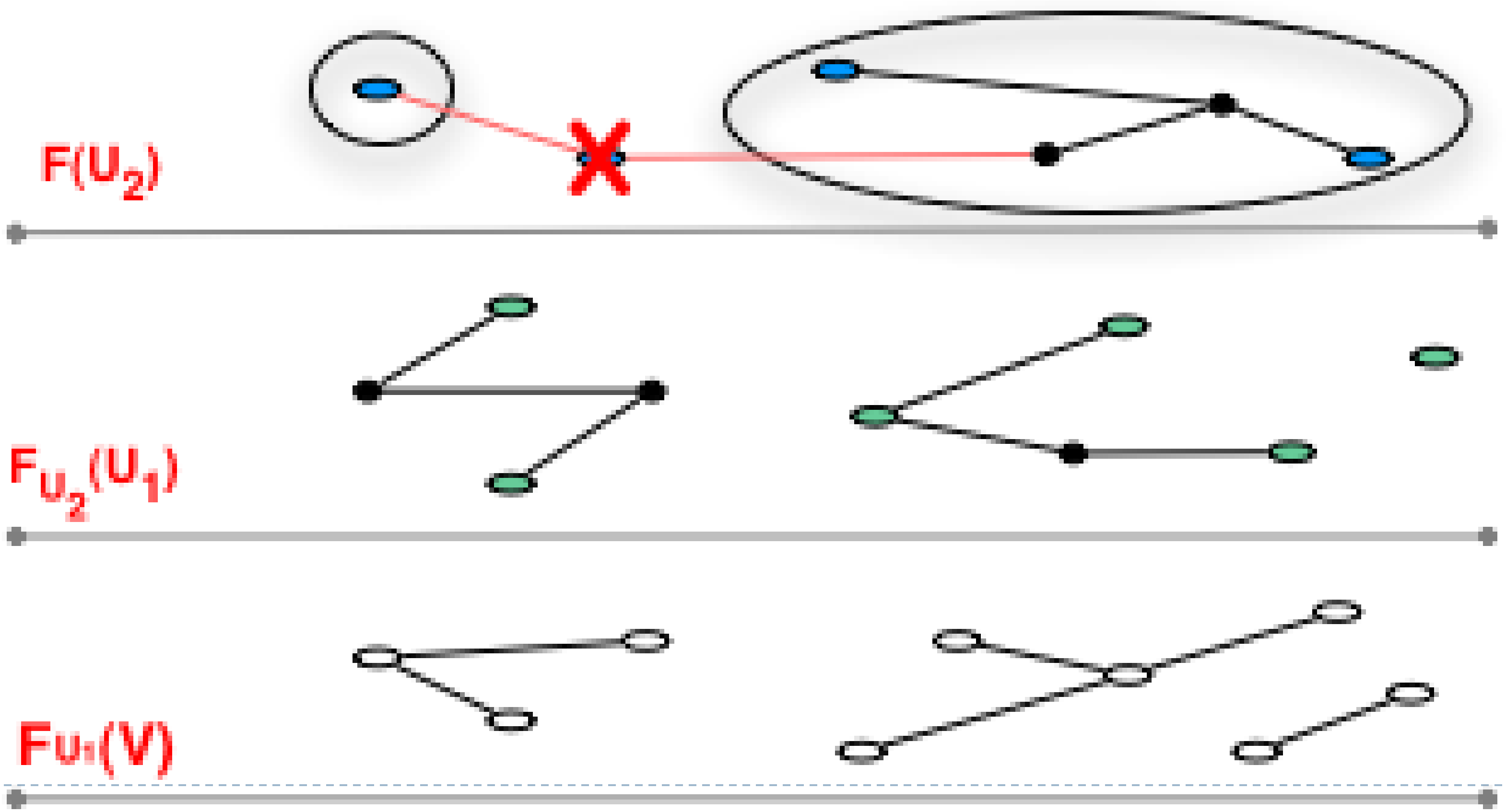


- These are the forests $F_{U1}(V)$, $F_{U2}(U_1)$, …, $F_{Up}(U_{p-1})$, $F(U_p)$.
- Every spanning forest may contain lower level vertices, but not higher level vertices.
- The spanning forests can reflect the connectivity through lower level vertices

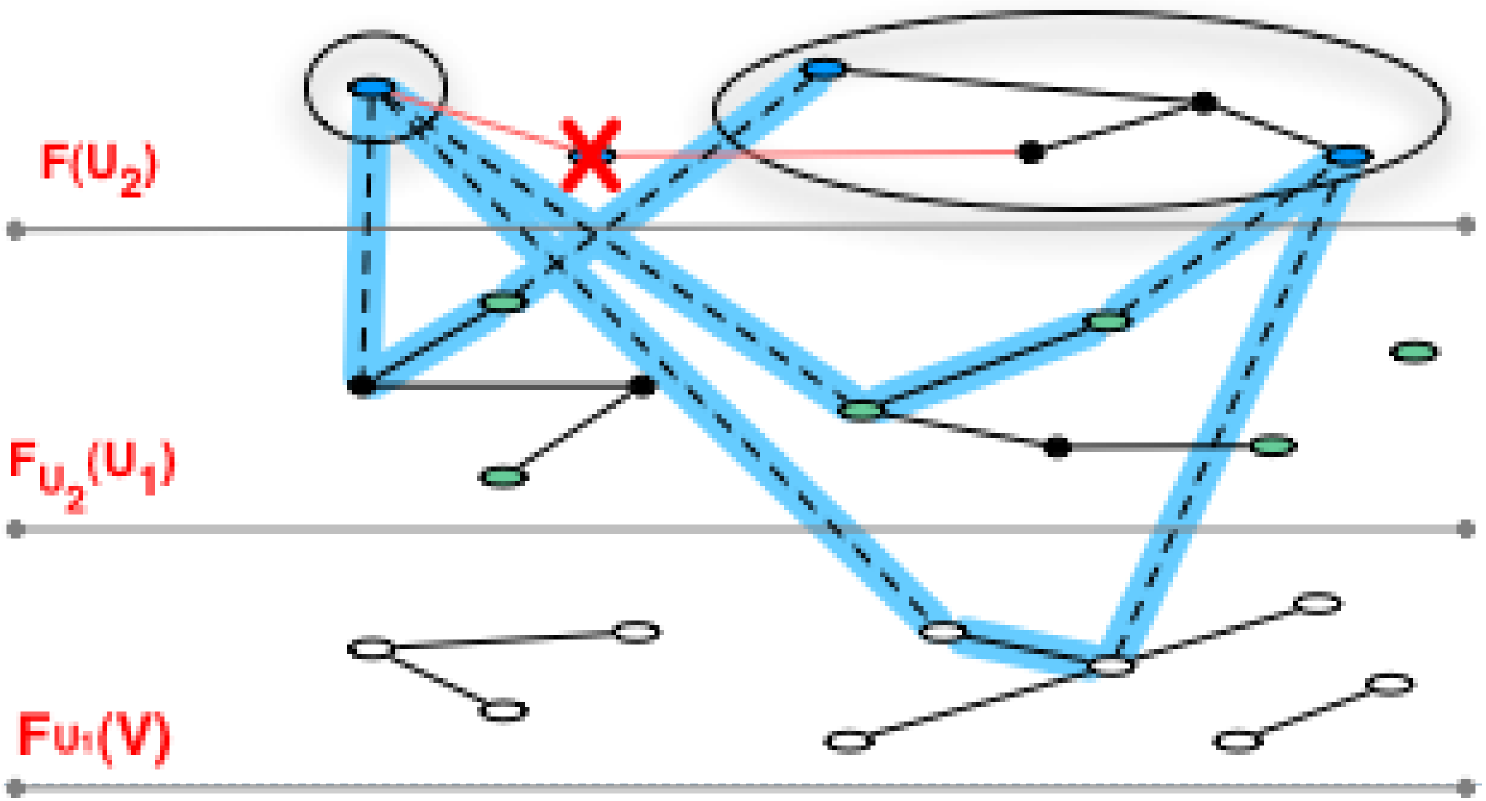


- Recall that $F_W(U)$= the forest connecting U\W in the subgraph G\W.

- When a vertex fails in a tree, we need to reconnect the subtrees split from it.

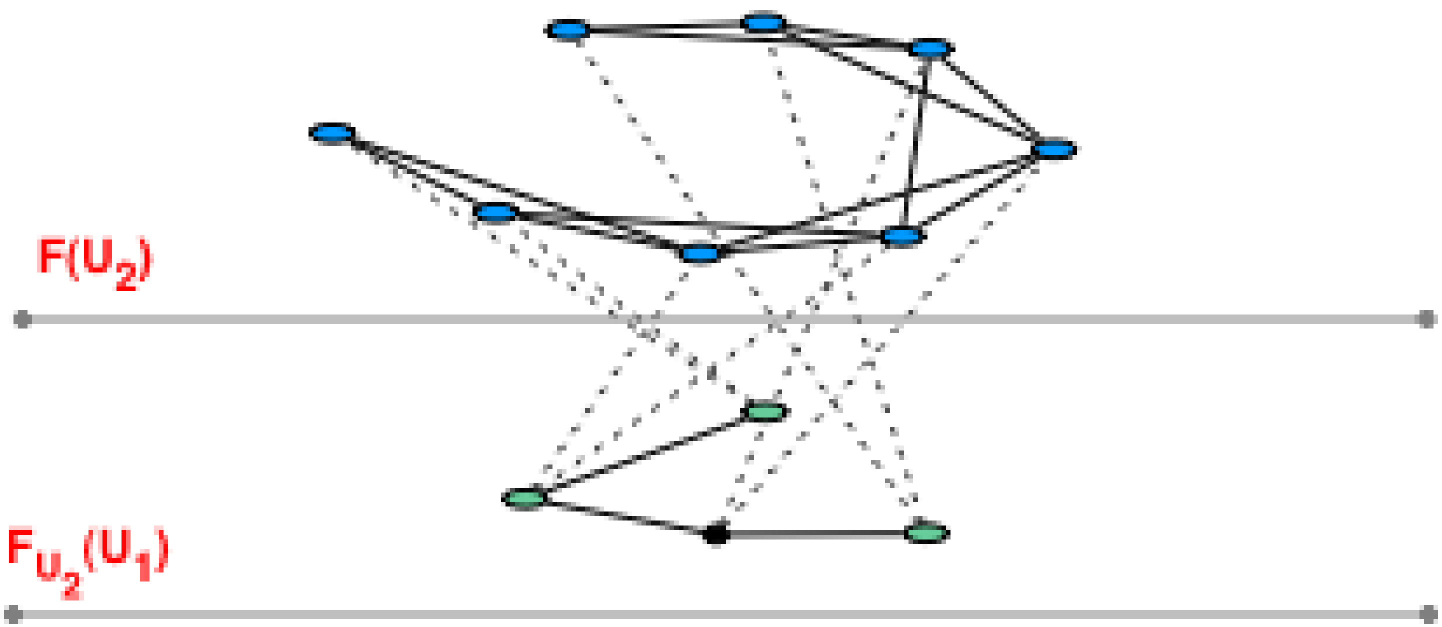


- The subtrees split from it may be connected by many trees of lower levels, the number of which is not bounded by poly(d).
- How to deal with this?



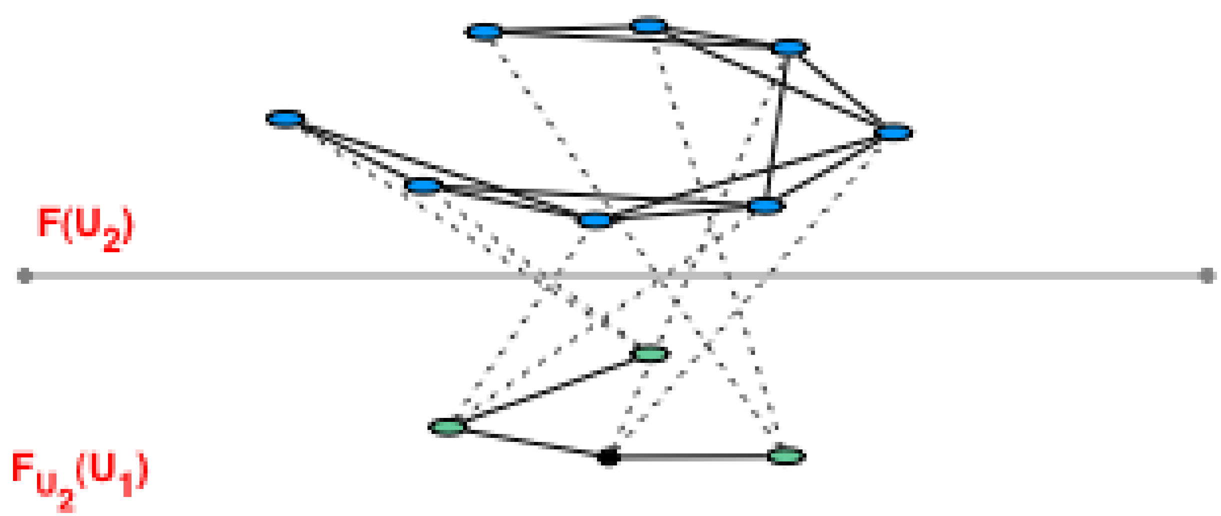
d-failure Graph

- Add artificial edges reflecting connectivity through lower level vertices.
- For the vertices $v_1, v_2, \ldots, v_n$ connected to a lower level tree ordered by the ET-tour of that tree, add edges (v_i, v_j) if $|i-j| \le d+1$.



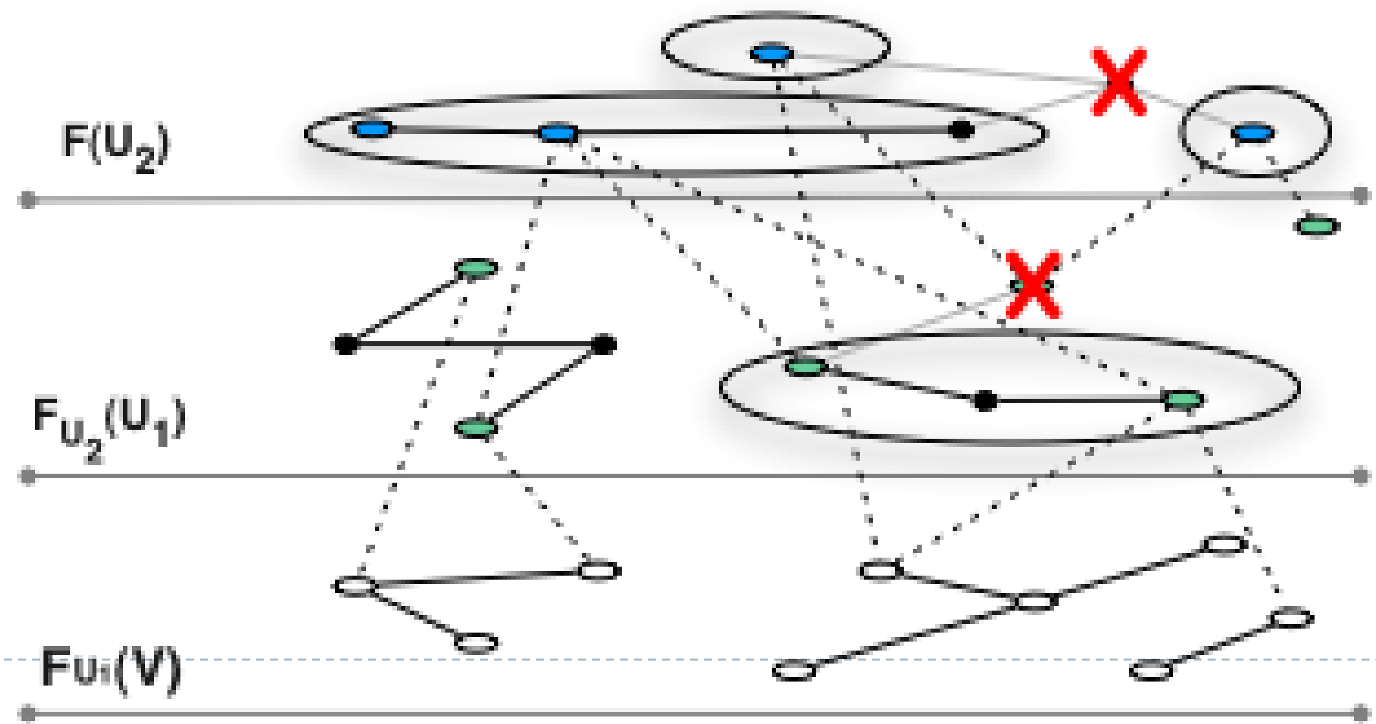
d-failure Graph

- Each vertex is adjacent to its 2(d+1) neighbors.
- Even when d vertices in the set fail, the graph on the active vertices is still connected.
- When the tree is split into two subtrees, we need to delete $O(d^2)$ edges.
- The space is O(d•m).



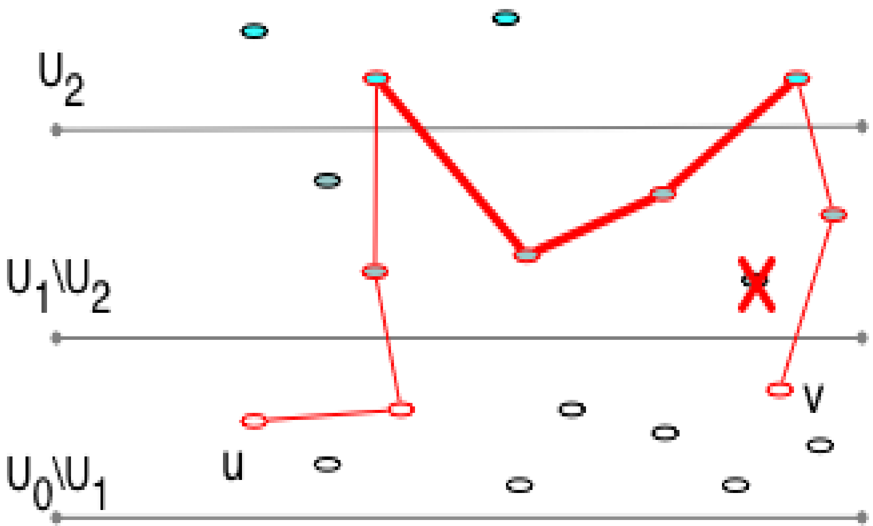
Processing d failures

- When d vertices fail, we will reconnect the spanning trees containing them.
  - By both original edges in G and artificial edges added by the d-failure graph.
  - The number of such subtrees is O(d•s•log n), so the time needed is its square $\tilde{O}(d^2 s^2)$.

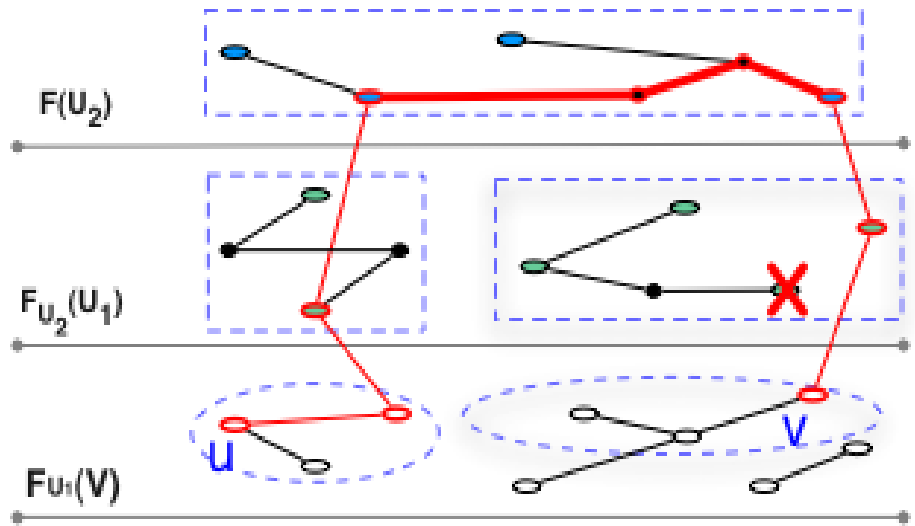


Answering a Query

- **A path connecting u and v may be like:**

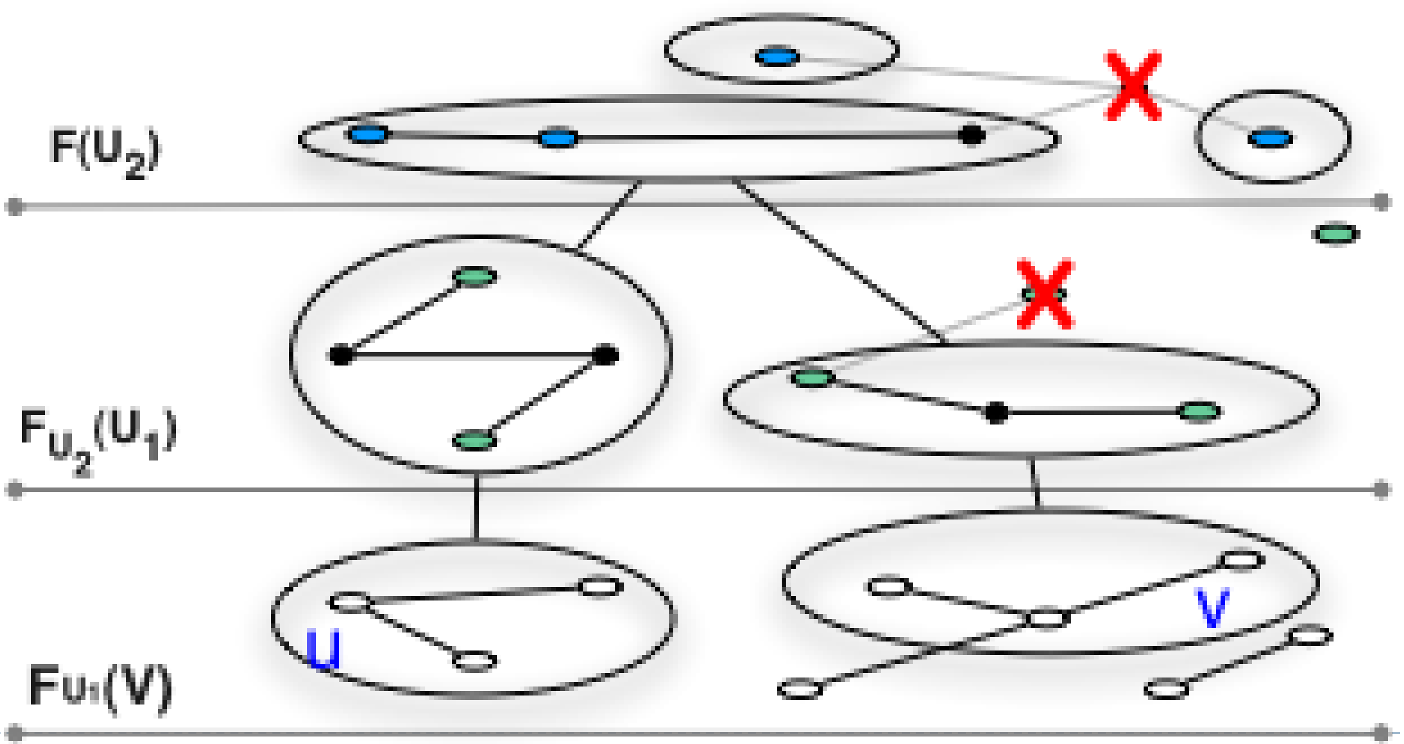


- **Consider the trees**



$F_W(U)$= the forest connecting U\W in the subgraph G\W.

- Consider the trees after reconnection. A tree can only be connected to one tree in every higher level forest.
- Check the connectivity between two vertices u and v:
  - Locate u and v in the forests
  - Find all the trees in higher levels connecting to the trees containing u and v.



Tradeoff between time and space

- Let $s=O(d^{c+1})$
- Processing time for d failures: $\tilde{O}(d^2s^2) = \tilde{O}(d^{2c+4})$.
- Query time: $O(d)$
  - Since in the reconnected components, we need to find a component other than the d failed vertices.
- Space: $\tilde{O}(d\square m\square n^{1/c})$
  - $\tilde{O}(n^{1/c})$ nodes in the hierarchy.
  - $\tilde{O}(dm)$ space per path.

Algorithms Overview

- d-failure model:
  - d-failure connectivity
- Real dynamic subgraph model:
  - Worst-case connectivity

Difficulties

- Turning a vertex "off" may split the graph into O(n) components.
  - We can't even spend O(1) time for every edge in the worst-case scenario.
  - The best worst-case edge update connectivity structure takes $O(n^{1/2})$ time per edge update.

Basic Ideas

- Partition the vertices into different sets by their degrees.
- Maintain the subgraph on theses sets differently:
  - Subgraph on low-degree vertices: use the dynamic connectivity with $O(n^{1/2})$ worst-case edge update time.
  - Subgraph on high-degree vertices: run a BFS in every update, since the number of vertices of degree $\geq k$ is bounded by $O(m/k)$.
- Add artificial edges to high-degree vertices to reflect the connectivity through low-degree vertices.

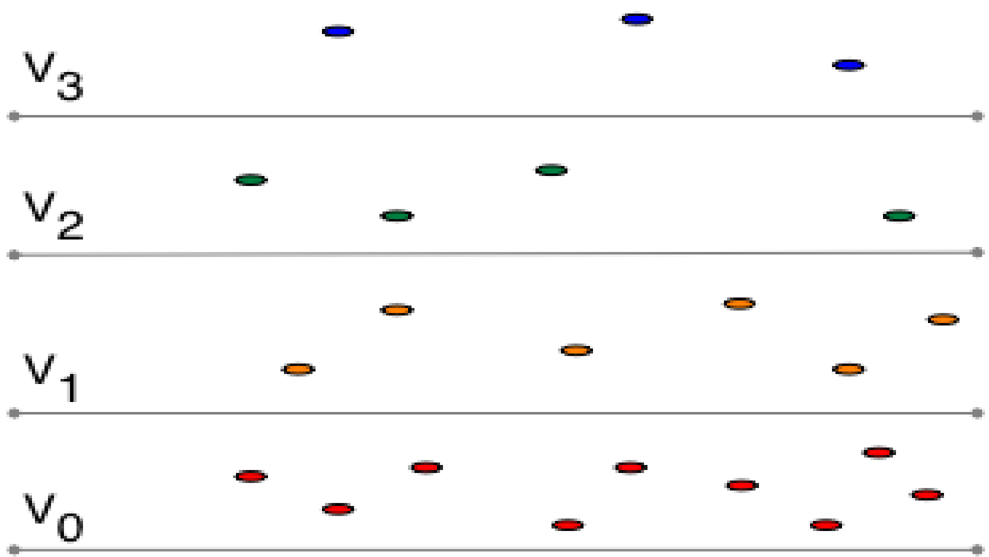
Simpler solution- $\tilde{O}(m^{0.9})$ Worst-case Update Time

- Partition the vertex set V (both "on" and "off" vertices) into 4 subsets by the degrees of vertices:

Subsets	Degree bounds	Size
V_0	$[1, m^{0.4})$	$O(m)$
V_1	$[m^{0.4}, m^{0.6})$	$O(m^{0.6})$
V_2	$[m^{0.6}, m^{0.9})$	$O(m^{0.4})$
V_3	$[m^{0.9}, m]$	$O(m^{0.1})$

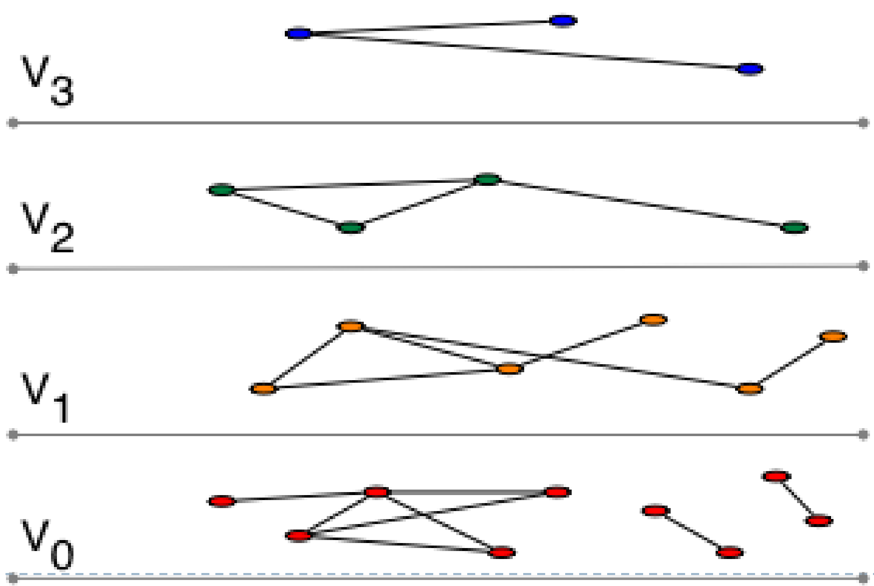
- Notice that these sets are static.

Subsets	Degree bounds	Size
V_0	$[1, m^{0.4})$	$O(m)$
V_1	$[m^{0.4}, m^{0.6})$	$O(m^{0.6})$
V_2	$[m^{0.6}, m^{0.9})$	$O(m^{0.4})$
V_3	$[m^{0.9}, m]$	$O(m^{0.1})$



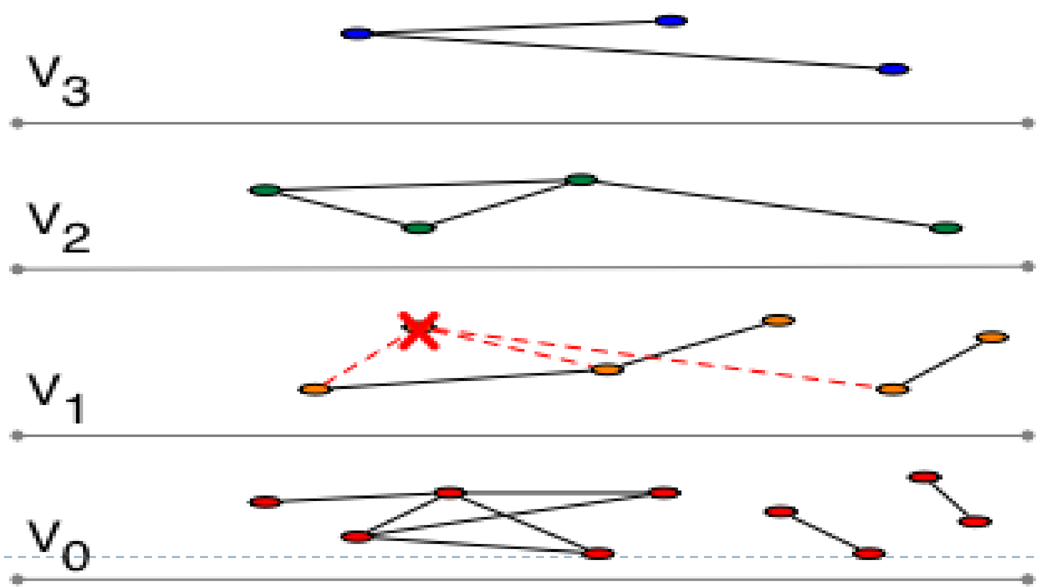
Subsets	Degree bounds	Size
V_0	$[1, m^{0.4})$	$O(m)$
V_1	$[m^{0.4}, m^{0.6})$	$O(m^{0.6})$
V_2	$[m^{0.6}, m^{0.9})$	$O(m^{0.4})$
V_3	$[m^{0.9}, m]$	$O(m^{0.1})$

For the subgraph of G induced by every subset, we need to maintain the connectivity dynamically.



If we use $O(n^{1/2})$ worst-case edge update connectivity oracle on the subgraph on V_0 and V_1, the update time will be:

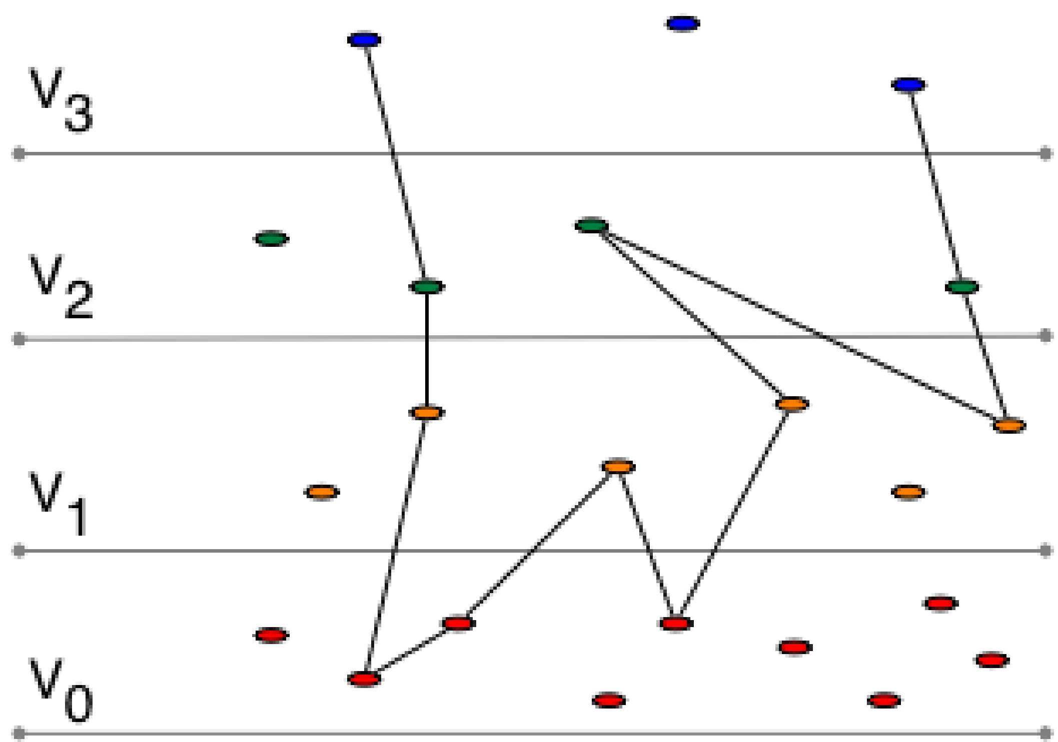
Subsets	Degree bounds	Size	Update Time ($Size^{0.5} \times Degree$)
V_0	$[1, m^{0.4})$	$O(m)$	$O(m^{0.5} \times m^{0.4}) = O(m^{0.9})$
V_1	$[m^{0.4}, m^{0.6})$	$O(m^{0.6})$	$O(m^{0.3} \times m^{0.6}) = O(m^{0.9})$
V_2	$[m^{0.6}, m^{0.9})$	$O(m^{0.4})$	
V_3	$[m^{0.9}, m]$	$O(m^{0.1})$	



We can just keep the subgraph of V_2 and V_3 and run a BFS on it after an update, which will take $O(m^{0.8})$ time.

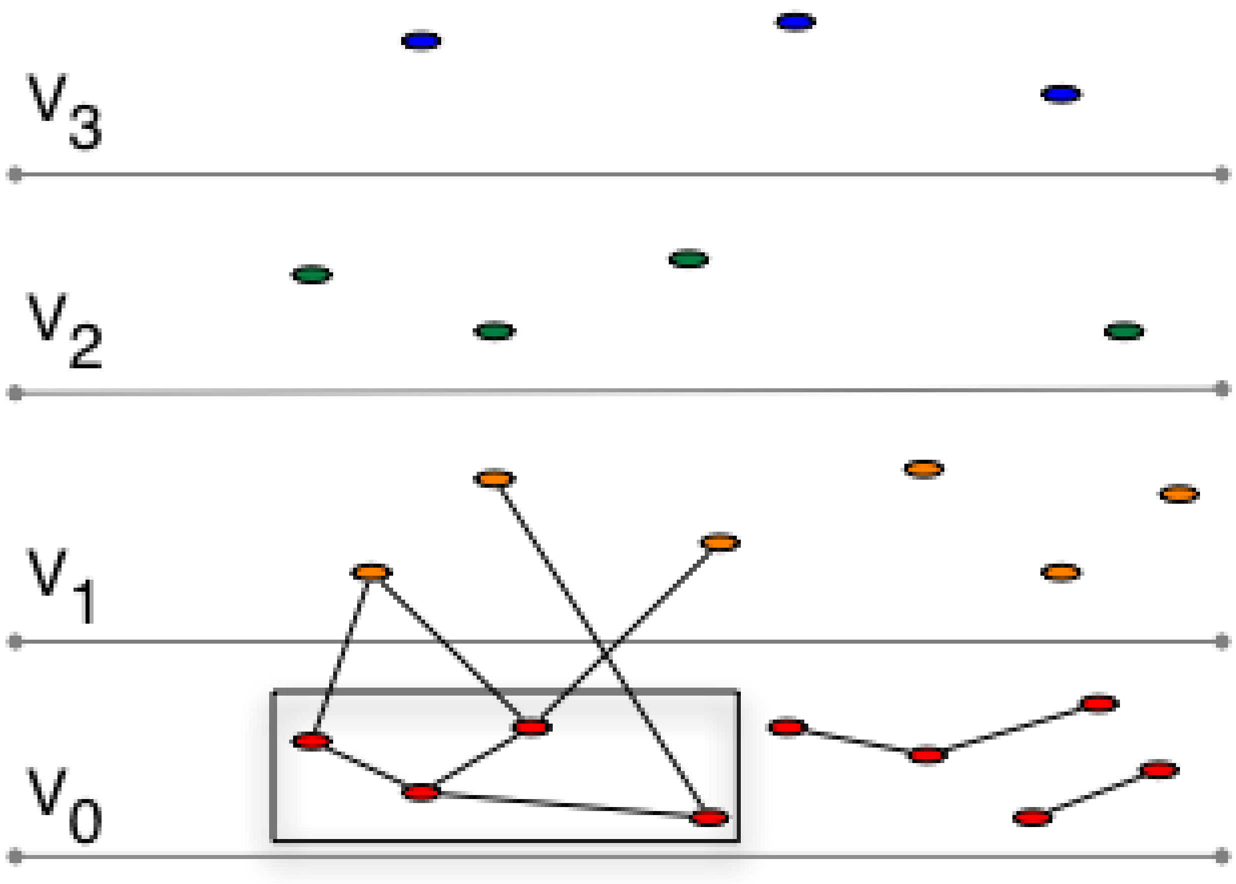
Subsets	Degree bounds	Size	Update Time
V_0	$[1, m^{0.4})$	$O(m)$	$O(m^{0.5} \times m^{0.4}) = O(m^{0.9})$
V_1	$[m^{0.4}, m^{0.6})$	$O(m^{0.6})$	$O(m^{0.3} \times m^{0.6}) = O(m^{0.9})$
V_2	$[m^{0.6}, m^{0.9})$	$O(m^{0.4})$	$O(m^{0.8})$
V_3	$[m^{0.9}, m]$	$O(m^{0.1})$	

However, a path connecting two vertices may be like this…

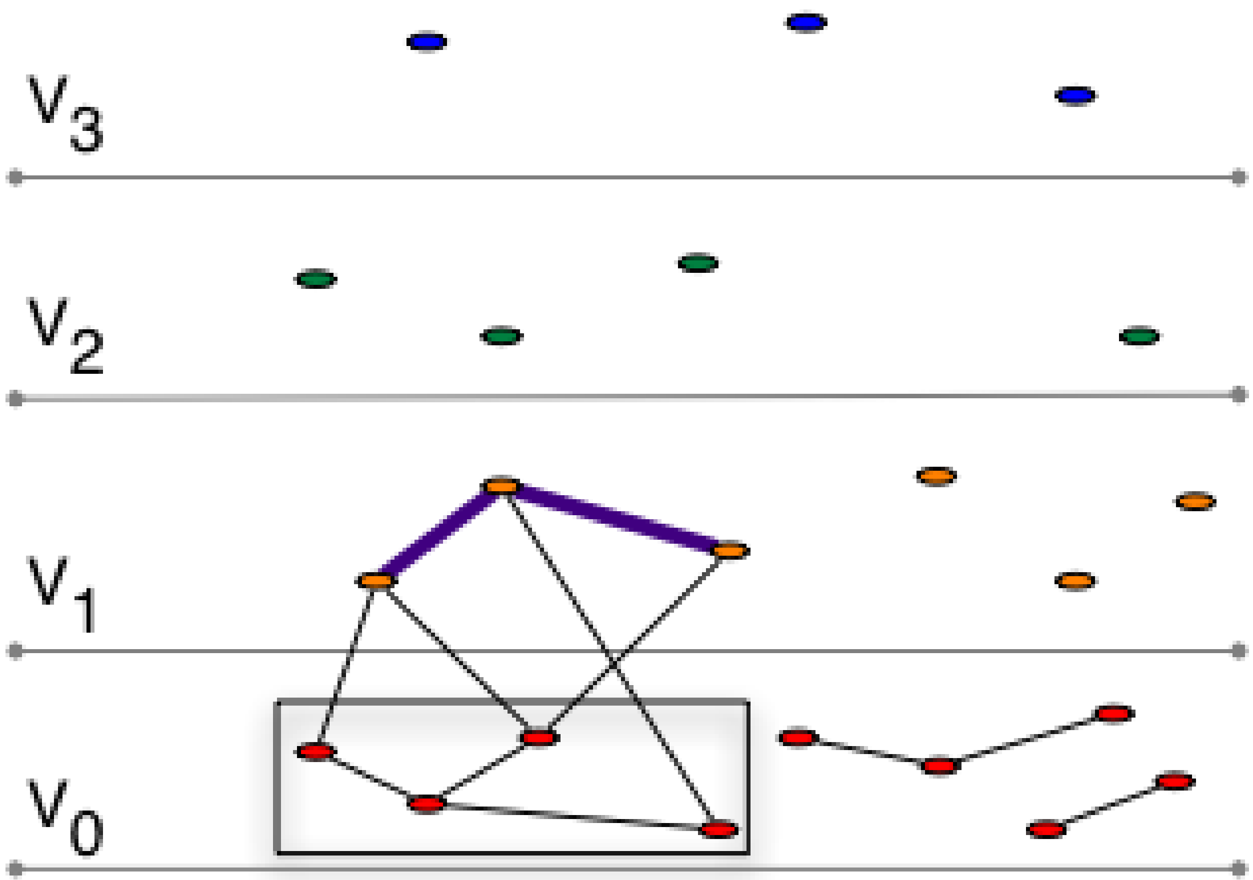


How to deal with these inter-set edges?

- Suppose there are some vertices of V_1 which are adjacent to the same connected component of V_0.

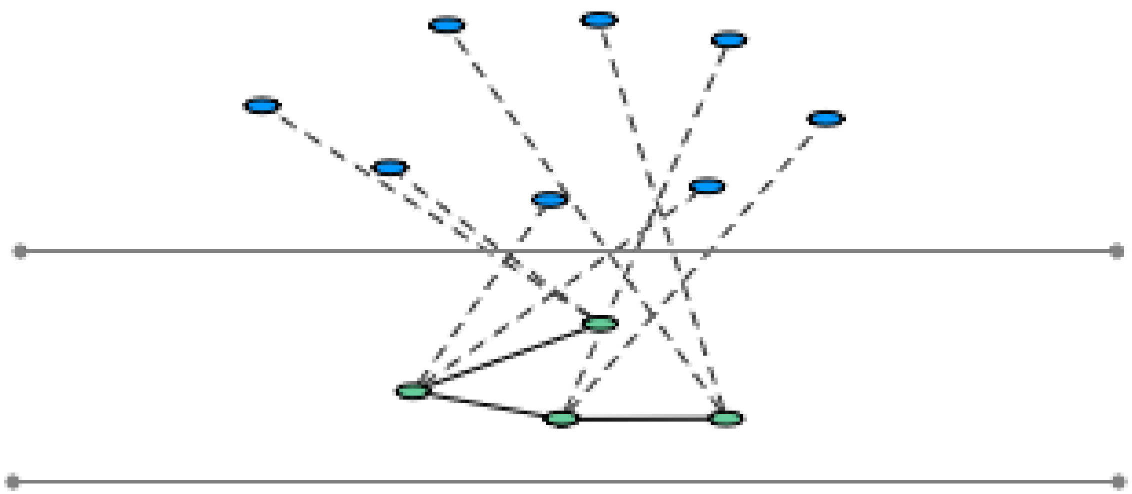


- Add artificial edges to connect these V_1 vertices.



Adjacency Graph

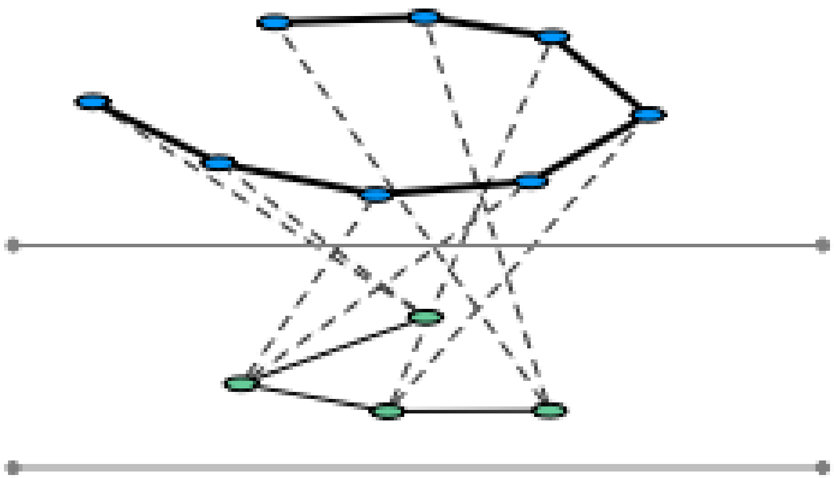
- Set of artificial edges maintaining the connectivity of high-level vertices through low-level vertices.



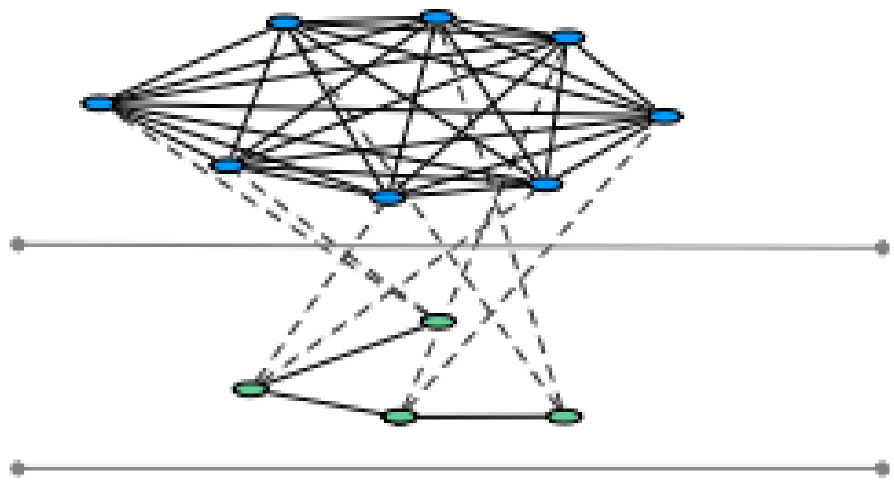
- Two types: Path graph and Complete graph.

Adjacency Graph

Path:

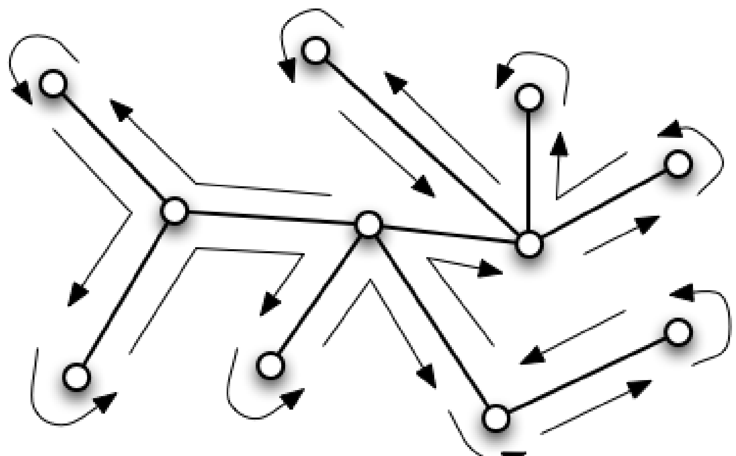


Complete Graph:



Euler Tour

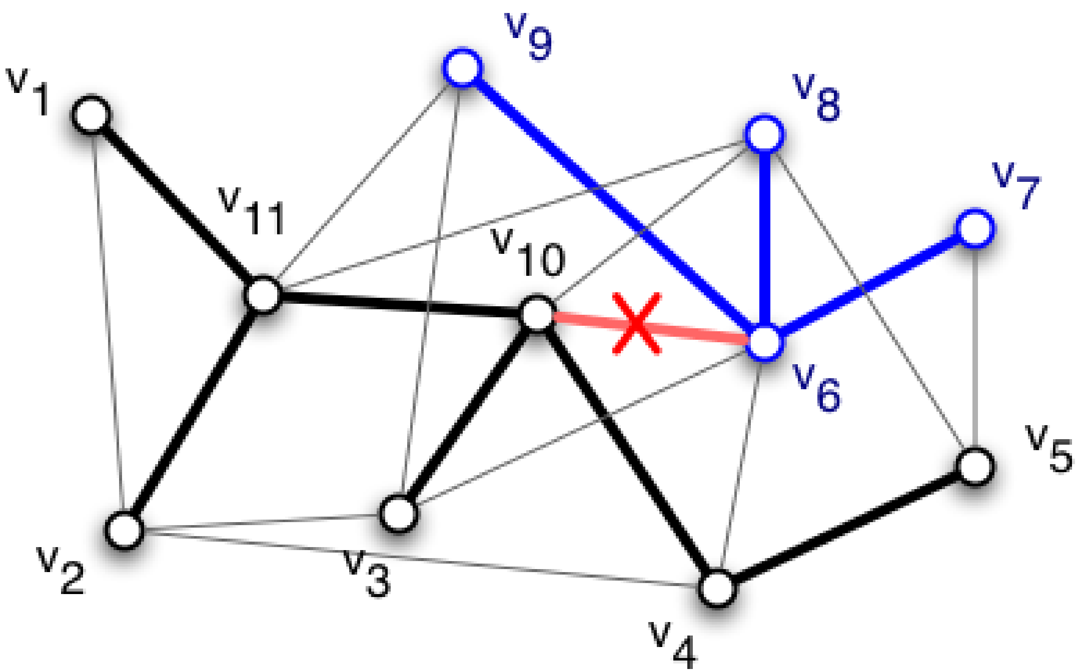
- Euler Tour of T:



- Every vertex can appear many times in the Euler Tour, but we only keep any one of them for each vertex to form a ET-list:

  $v_1, \quad v_2, \quad \ldots \; v_n$

When we delete a tree edge, the ET-list will be divided into ≤3 parts, and we need to merge two lists.

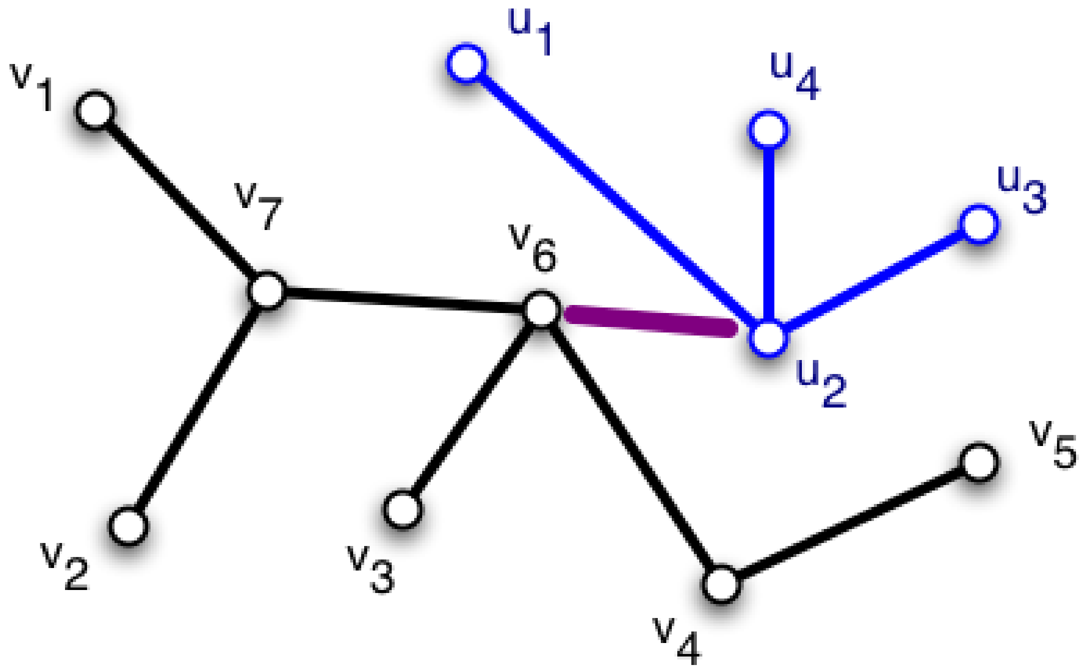


$v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}$

$\Downarrow$

$(v_1, v_2, v_3, v_4, v_5, v_{10}, v_{11})$; (v_6, v_7, v_8, v_9)

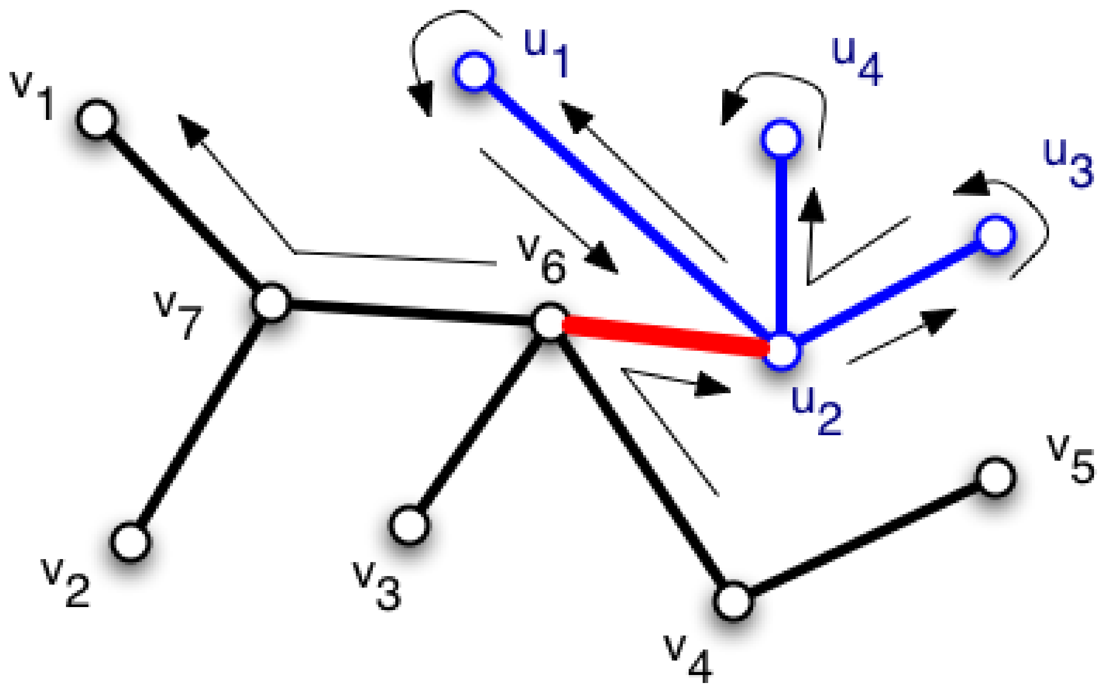
When we connect two trees by an edge, we need to split the ET-lists of the two trees from the vertices on that edge …



$(v_1, v_2, v_3, v_4, v_5), (v_6, v_7)$

$(u_1), (u_2, u_3, u_4)$

When we connect two trees by an edge, we need to split the ET-lists of the two trees from the vertices on that edge, and merge them in the right order.



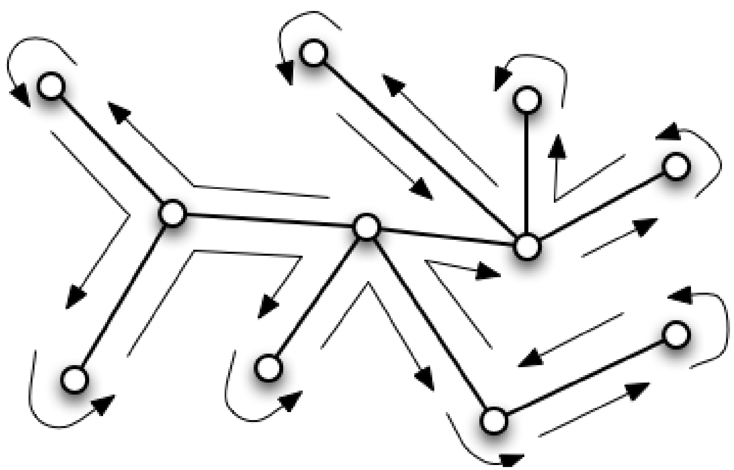
$(v_1, v_2, v_3, v_4, v_5), (v_6, v_7); (u_1), (u_2, u_3, u_4)$

$\Downarrow$

$(v_1, v_2, v_3, v_4, v_5, u_2, u_3, u_4, u_1, v_6, v_7)$

Euler Tour

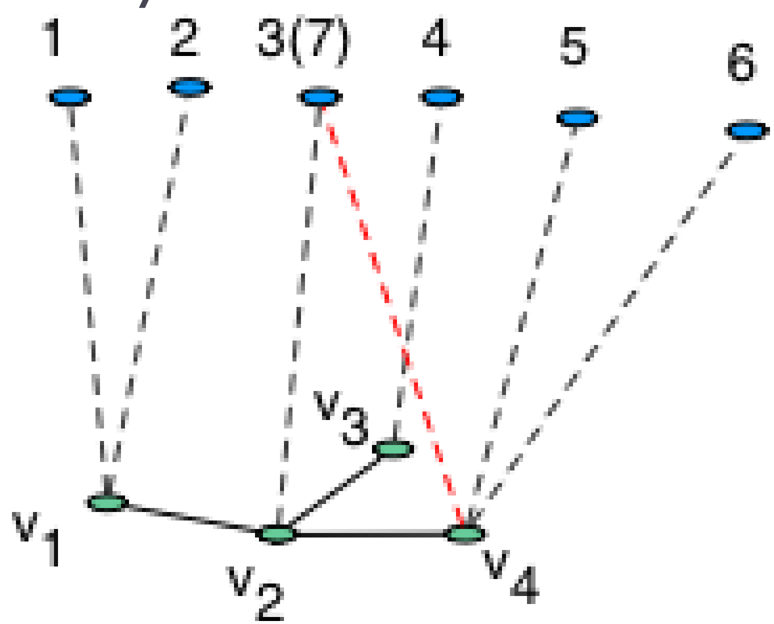
- Euler Tour of T:



- So we only need O(1) link & cut operations to maintain the ET-lists per tree merging or splitting.

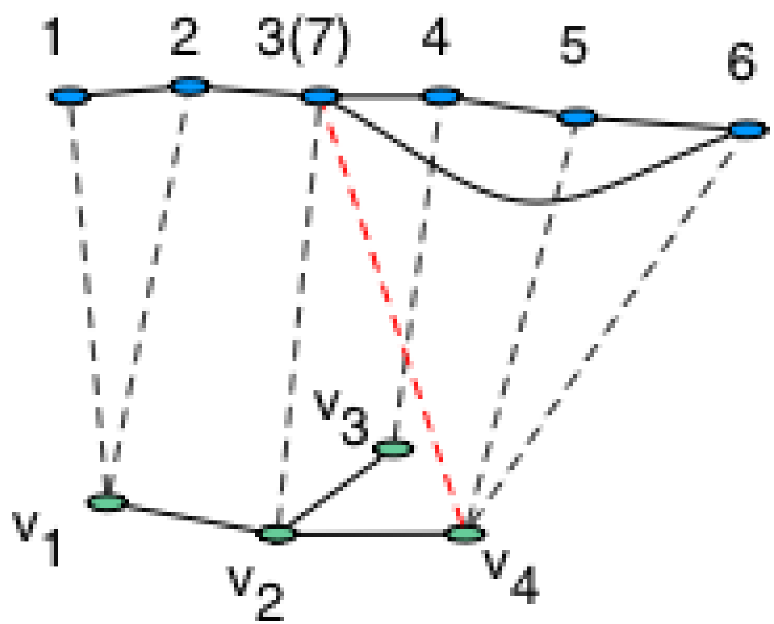
Path Graph

- Find the ET-list of the spanning tree in low-level.
- Order its adjacent **"on"** vertices on high-level by the ET-list
  - Notice that a vertex can appear multiple times since it may be adjacent to many vertices in low-level.



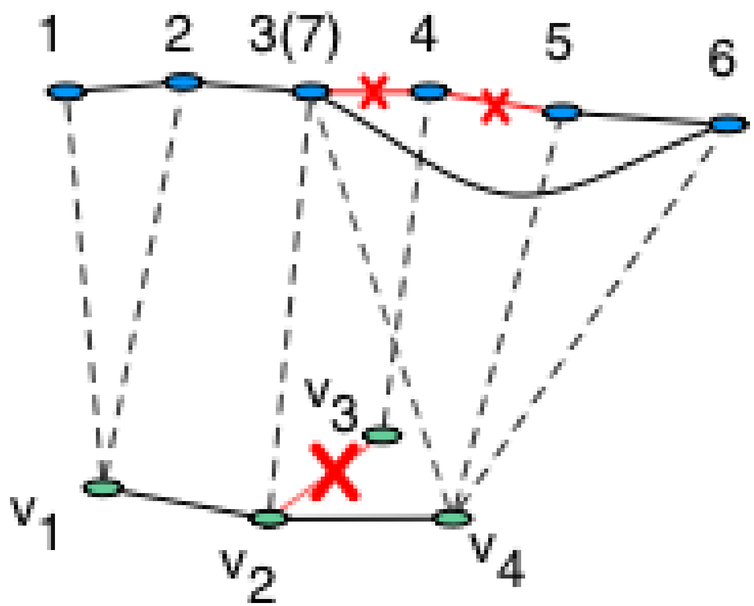
Path Graph

- Find the ET-list of the spanning tree in low-level.
- Order its adjacent **"on"** vertices on high-level by the ET-list
- Then connect them by a path in this order



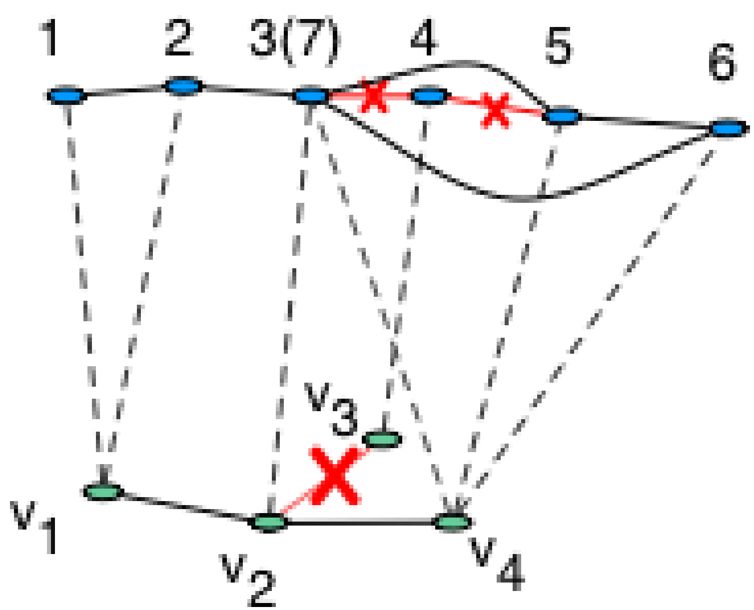
Merge or split trees

- When we delete a tree edge, since the ET-list will be split into at most 3 parts, the path graph will also be split into ≤3 parts.

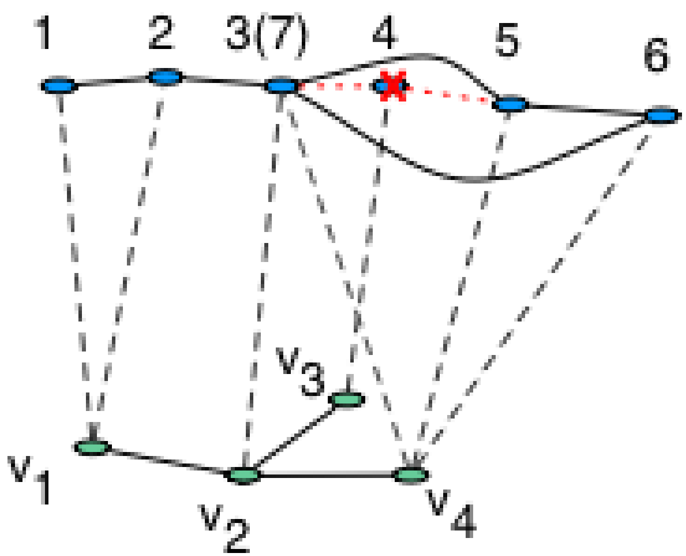


Merge or split trees

- Then we need to reconnect the path.
- Similar to tree merging. So both merging and splitting by one edge will need O(1) links/cuts.

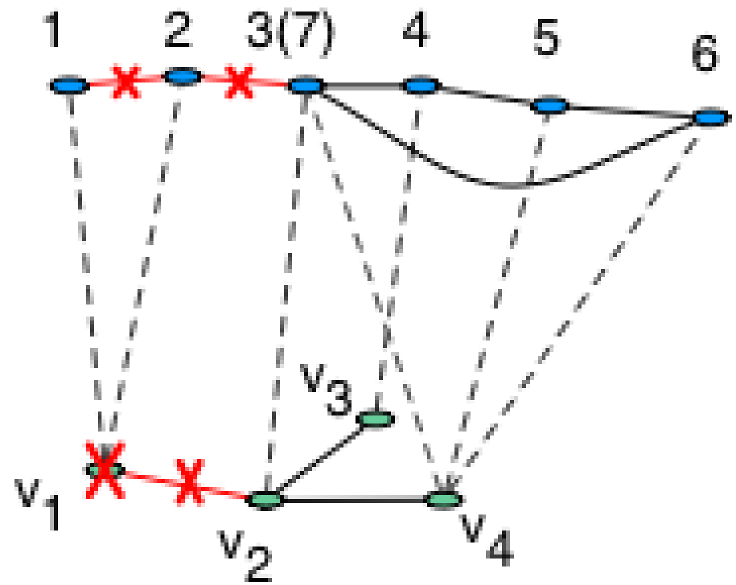


When we update a high-level vertex,



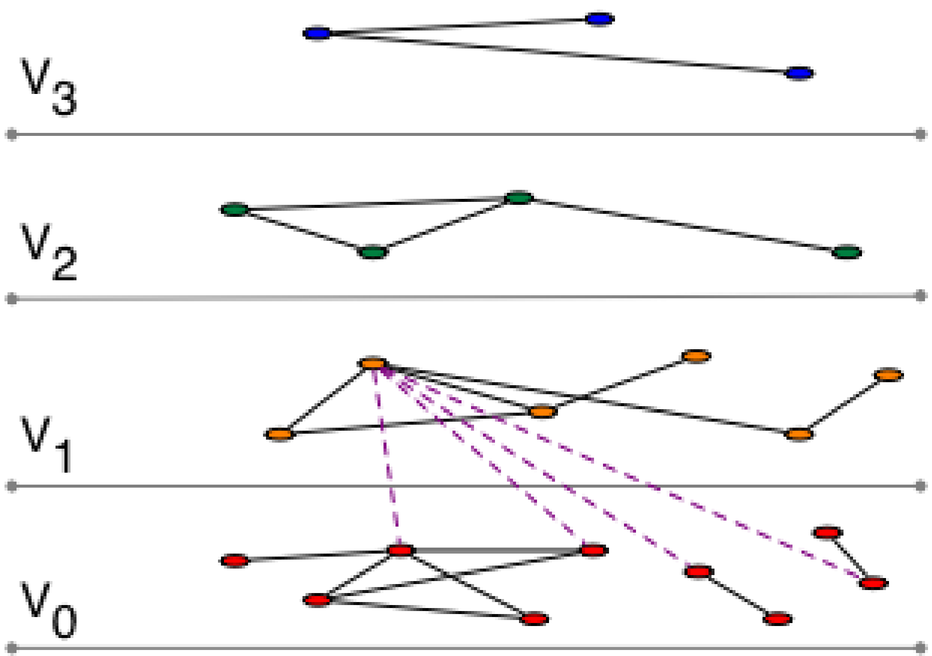
We need to update O(1) edges in the path for every vertex in low-level it is adjacent to.

When we update a low-level vertex,



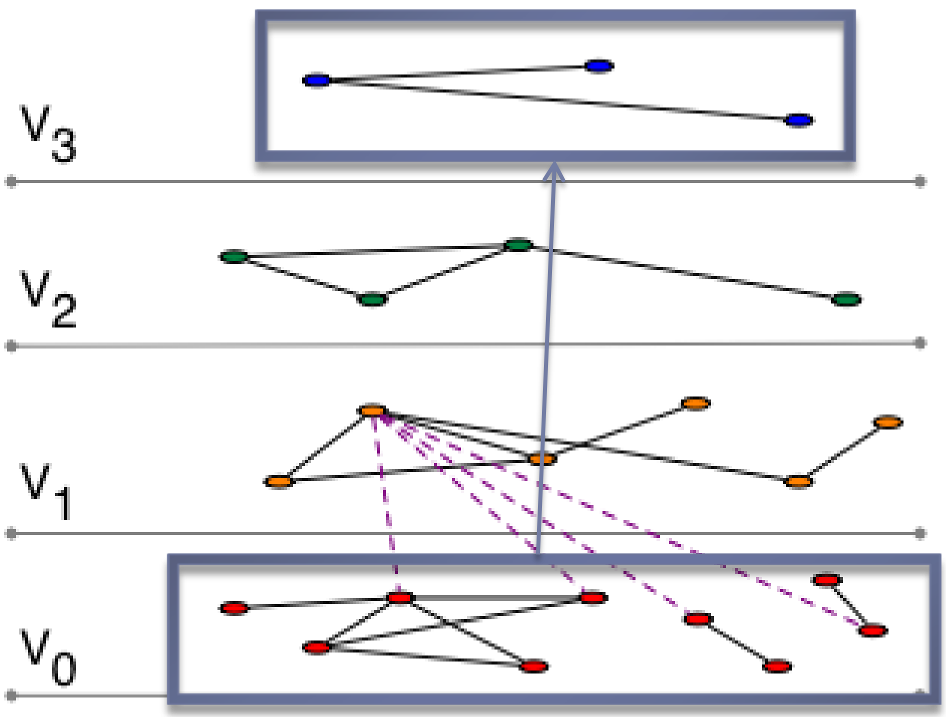
1. Maintain the spanning forests in low-level.
2. Update its adjaceny vertices in high-level.

- A high-level vertex may be adjacent to many trees in low-level.
- So the time needed to update the path graph is linear to the degree of the updated vertex.



Then how to deal with the highest level vertices without degree bound?

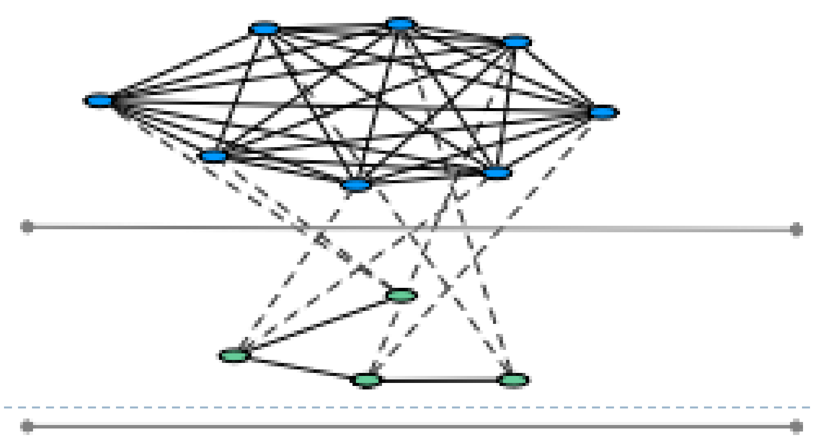
- So the time needed to update the path graph is linear to the degree of the updated vertex.
- The number of trees in V_0 adjacent to a vertex in V_3 may be $\Theta(n)$.



Then how to deal with the highest level vertices without degree bound?

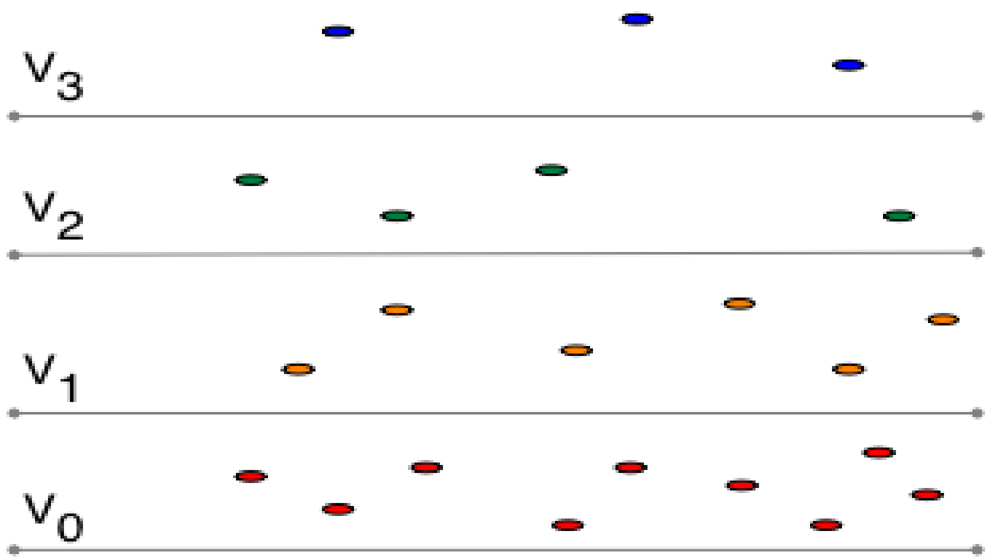
Complete Graph

- Connect every pair of vertices (both "on" and "off") by an edge.
- When we update a low-level vertex, re-compute the entire graph;
- When we update a high-level vertex, do nothing to this graph.
  - Since the remaining "on" vertices are still connected.



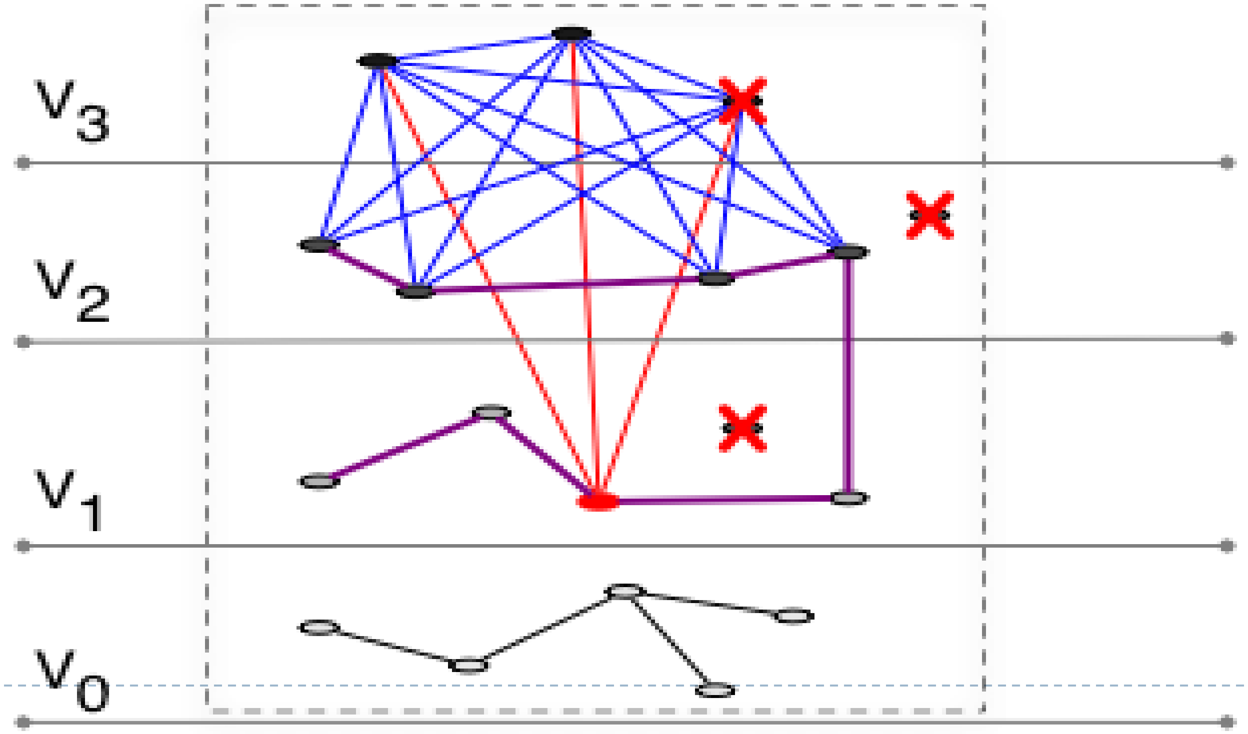
Recall the partition of vertices by degrees.

Subsets	**Degree bounds**	**Size**
V_0	$[1, m^{0.4})$	$O(m)$
V_1	$[m^{0.4}, m^{0.6})$	$O(m^{0.6})$
V_2	$[m^{0.6}, m^{0.9})$	$O(m^{0.4})$
V_3	$[m^{0.9}, m]$	$O(m^{0.1})$



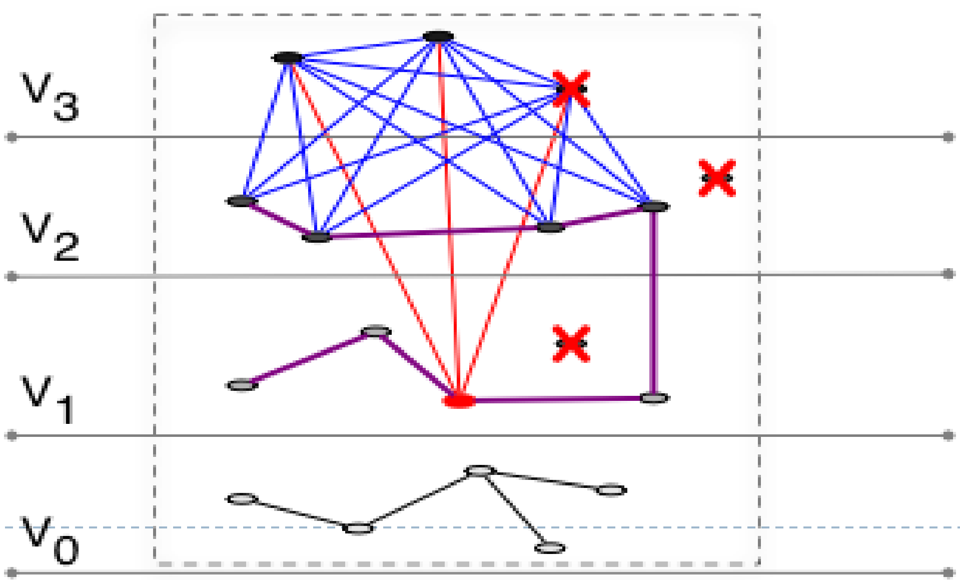
The structure

- Consider the vertices adjacent to a spanning tree in V_0.
  - V_1 and V_2: path graph;
  - V_2 to V_3 and within V_3: complete graph;
  - V_1 to V_3: arbitrarily choose an active vertex in V_1 and connect it to all vertices in V_3.



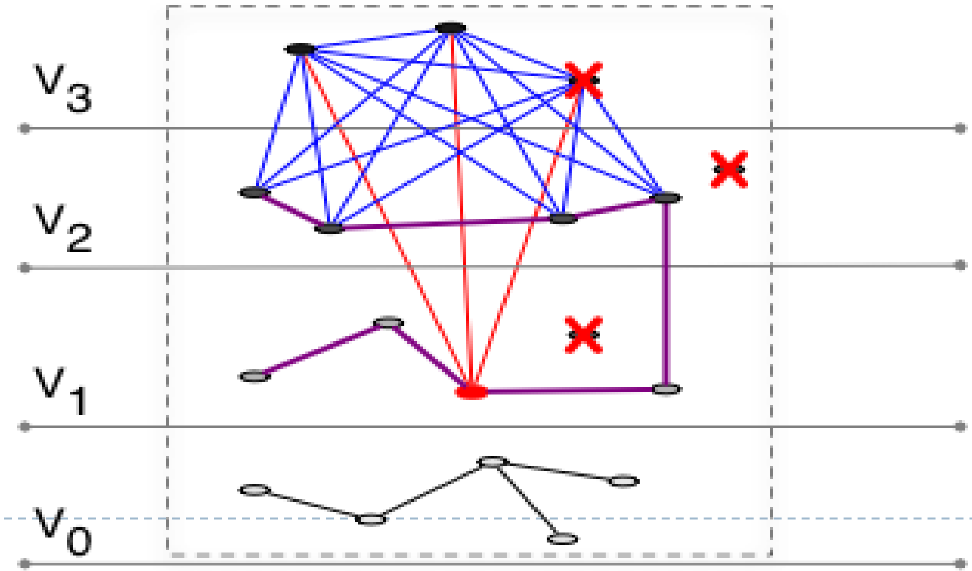
Update Time:

- V_1 and V_2: path graph;
  - Update V_0: changes $O(m^{0.4})$ edges, takes $O(m^{0.9})$ time.
  - Update V_1: changes $O(m^{0.6})$ edges in V_1 and V_2, takes $O(m^{0.9})$ time.
  - Update V_2: changes $O(m^{0.9})$ edges in V_2, we just keep those edges. (Without using a dynamic structure)



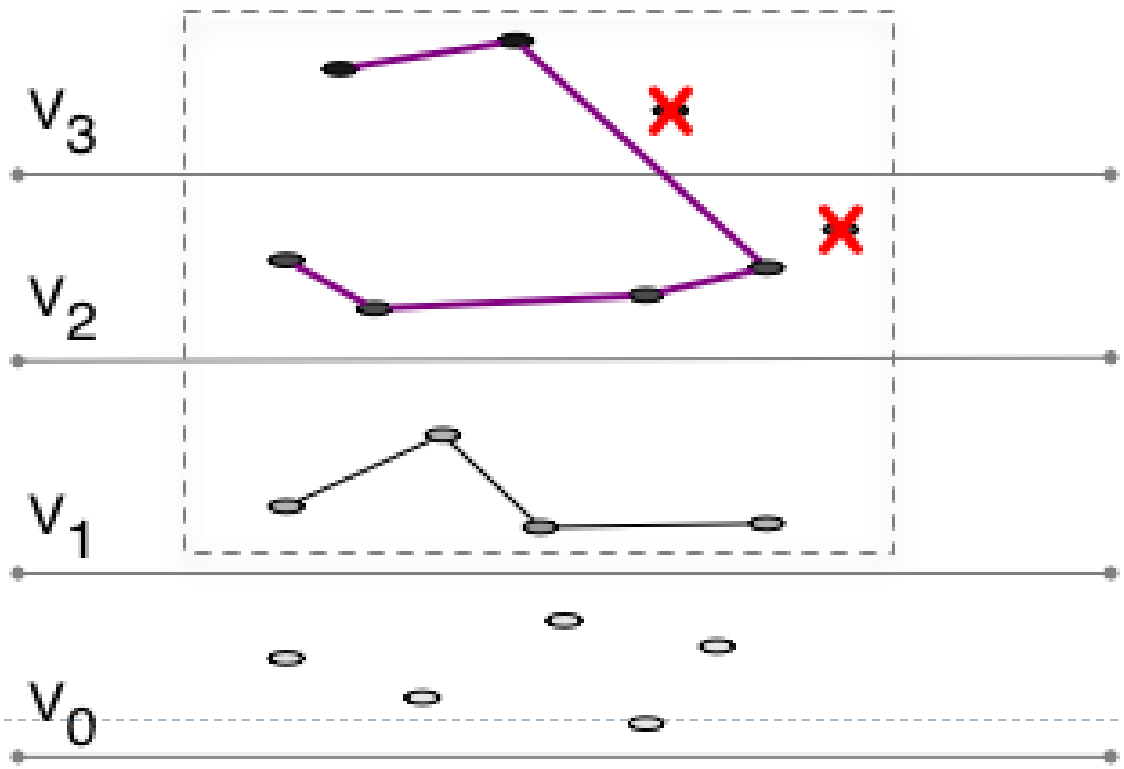
Update Time:

- V_1 and V_2: path graph;   (Time bound still $O(m^{0.9})$)
- V_2 to V_3 and within V_3: complete graph;
  - Update a vertex in V_0 will change $O(m^{0.4})$ tree edges, each will change $|V_2|\times|V_3|=O(m^{0.5})$ edges.
- V_1 to V_3: arbitrarily choose an active vertex in V_1 and connect it to all vertices in V_3.
  - degree(V_1) $\times|V_3|=O(m^{0.5})$ edges



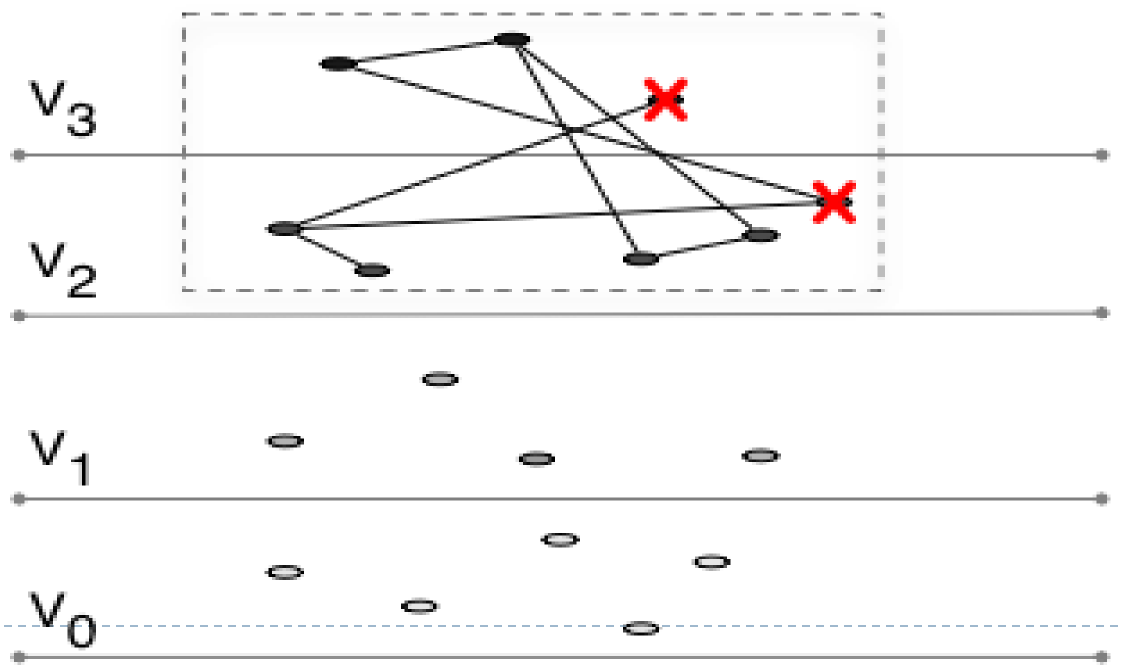
The structure

- Consider the vertices adjacent to a spanning tree in V_1.
  - V_2 and V_3: path graph;
    - Degree from V_2 and V_3 to V_1 is bounded by $|V_1|=O(m^{0.6})$.



The structure

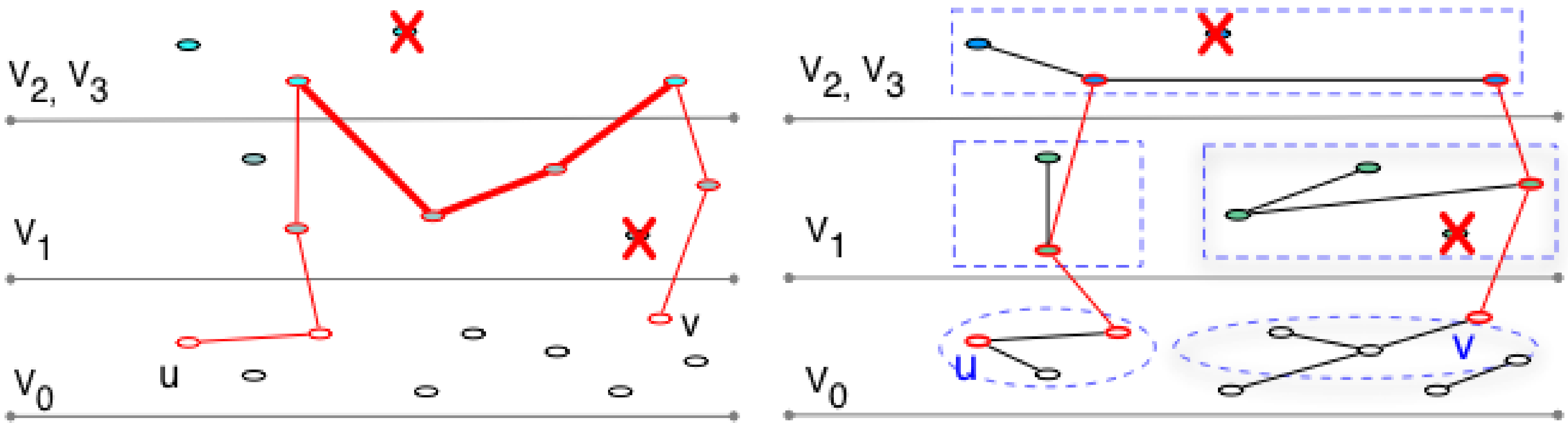
- For the vertices in V_2 and V_3, just keep the all the edges (original in G and artificial) on them, and run a BFS on the "on" vertices after an update.
  - It takes $(|V_2|+|V_3|)^2=O(m^{0.8})$ time.



Answering a Query

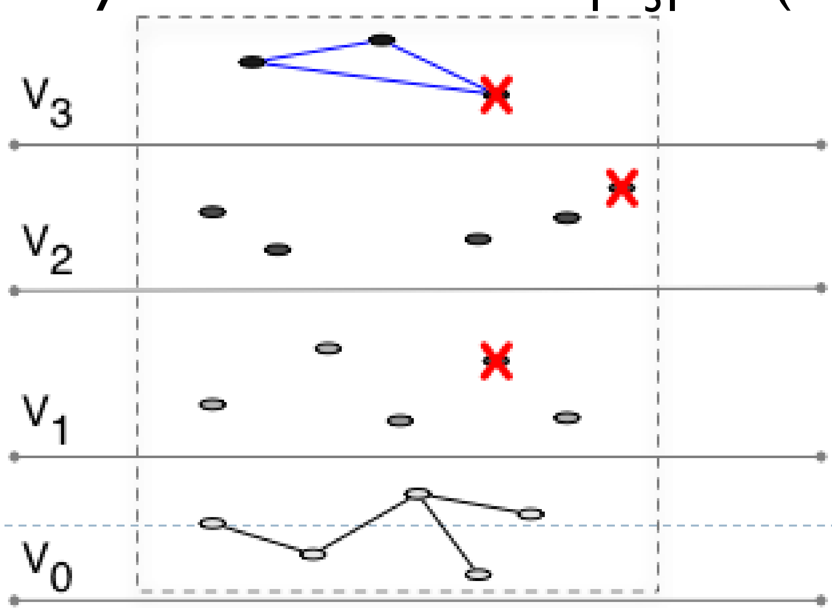
A path connecting u and v may be like:

We just need to find a common high-level spanning tree of them.



Query Time

- Finding a high-level "on" vertex in path graph only takes O(1) time.
- Since we use the complete graph from V_0 to V_3, we do not record the "on" vertices in V_3 adjacent to a tree in V_0 in the structure.
- So we need to check all the vertices in V_3 adjacent to the tree in V_0 whether they are "on". It takes $|V_3|=O(m^{0.1})$ time.



Reduce the update time to $\tilde{O}(m^{0.8})$

- Divide V into O(log n) sets:
  - Use the path graph on all of subsets of V_1.
  - New query time: $|V_3|=O(m^{0.2})$.

V_1 :degree $[m^{0.2}, m^{0.6})$,
Divided into O(log n)
subsets

Subsets	**Degree bounds**	**Size**
V_0	$[1, m^{0.2})$	$O(m)$
$V_{1,1}$	$[m^{0.2}, 2m^{0.2})$	$O(m^{0.8})$
	…	
$V_{1,i}$	$[2^i m^{0.2}, 2^{i+1} m^{0.2})$	$O(m^{0.8}/2^i)$
	…	
V_2	$[m^{0.6}, m^{0.8})$	$O(m^{0.4})$
V_3	$[m^{0.8}, m]$	$O(m^{0.2})$

Why $O(m^{0.8})$ Update Time?

- Worst-case edge update connectivity structure for low-degree vertices ($\leq k$):
  - Update time at least $O(k*(m/k)^{1/2})$
  - Need to make this degree bound precise.
- BFS for high-degree vertices ($>k$):
  - Update time: $(m/k)^2$

- Balance them: $k=m^{0.6}$, update time: $O(m^{0.8})$.

Open Problems

- Can we find subgraph connectivity oracle satisfying:
  - Query Time×Update Time=o(m).
- Or prove an $m^{\Omega(1)}$ lower bound.

- Dynamic subgraph reachability in directed graph?
- Multi-failure reachability in directed graph
  - We have a $\tilde{O}(n^2)$ space and O(log n) query time structure for dual-failure distance in directed graph [Duan & Pettie, 2009].

Homework and Exam

- Proposed oral exam time: 30.07-01.08
- Extra Assignment 12