# Interactive Large-Scale Data and Graph Analytics

Graph Analytics

**Oliver Alvarado Rodriguez**, Naren Khatwani, Zhihui Du, David Bader

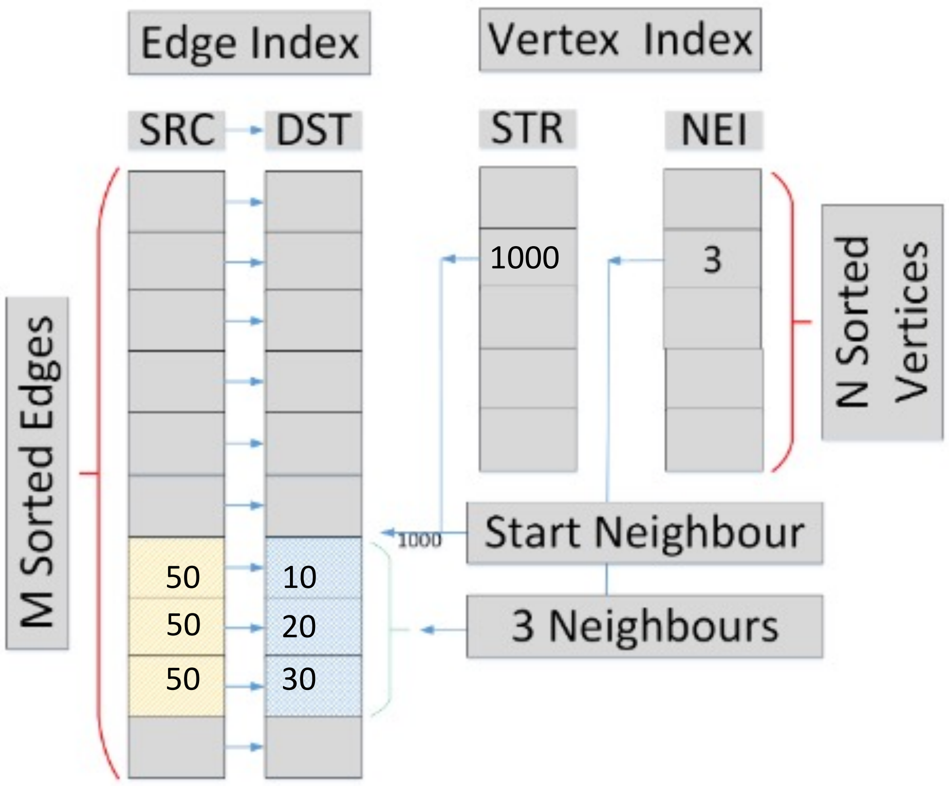
New Jersey Institute of Technology

# Outline

1. Explanation of the data structure.
2. Overview of our algorithmic graph theory.
3. Example of graph analytics from a Jupyter notebook.

# Arachne Double-Index (DI) Data Structure

[Alvarado Rodriguez, Du, Patchett, Li, Bader 2022]




**Advantages of DI over CSR:**

1. O(1) time complexity:
   - locating a vertex from a given edge ID.
   - locating the adjacency list from a given vertex ID.
2. We can search from edge ID to vertex ID, this is not possible in CSR.
3. DI can support both edge-centric and vertex-centric algorithms whereas CSR can only support the latter.
4. DI can easily achieve load balancing with the edge array being distributed equally amongst many locales.



New Jersey Institute of Technology

# Graph Algorithms in Arachne

- **Breadth-first search** [Du, Alvarado Rodriguez, Bader 2021]

  Returns an array of size $n$ with how many hops away some vertex $v$ is from an initial vertex $u$.

- **Connected components** [Du, Alvarado Rodriguez, Bader 2021]

  Returns an array of size $n$ where all vertices who belong to the same component have the same value $x$. The value of $x$ is the label of the largest vertex in the component.

- **Triangle counting** [Du, Alvarado Rodriguez, Patchett, Bader 2021]

  Returns the number of triangles in a graph.

- **Truss Analytics** [Du, Patchett, Bader 2021][Du, Patchett, Alvarado Rodriguez, Li, Bader 2022]

  K-truss returns every edge in the truss where each edge must be a part of $k-2$ triangles that are made up of nodes in that truss. Max truss returns the maximum $k$. Truss decomposition returns the maximum $k$ for each edge.

- **Triangle centrality** [Patchett, Du, Bader 2022][Patchett, 2022]

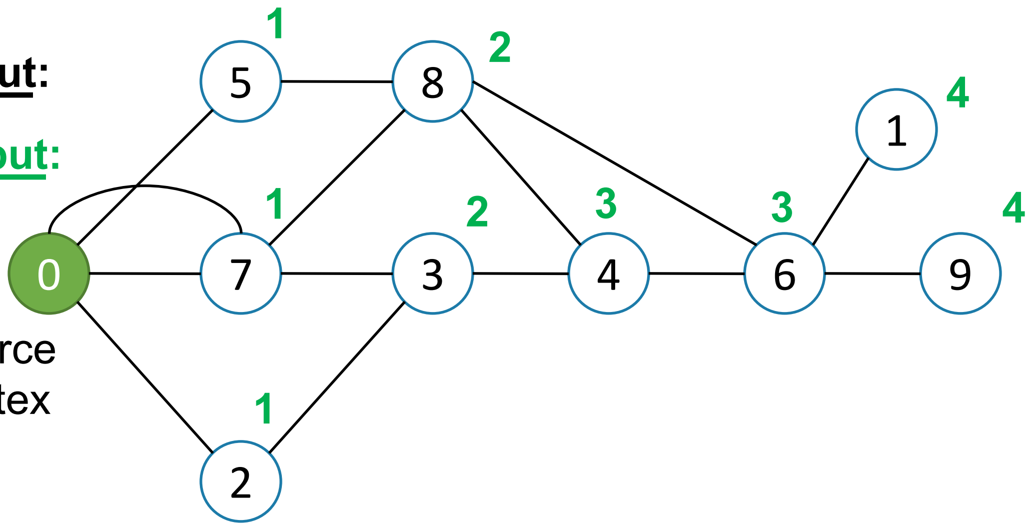
  Returns an array of size $n$ with the proportion of triangles centered at a vertex $v$.

NJIT
New Jersey Institute of Technology

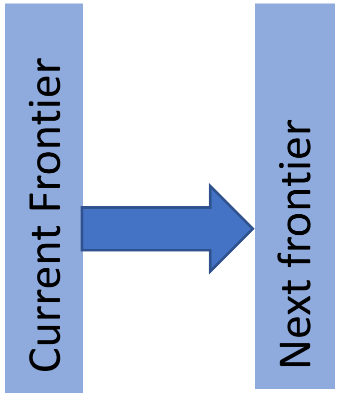
# Parallel Breadth-First Search Example

**Input:**

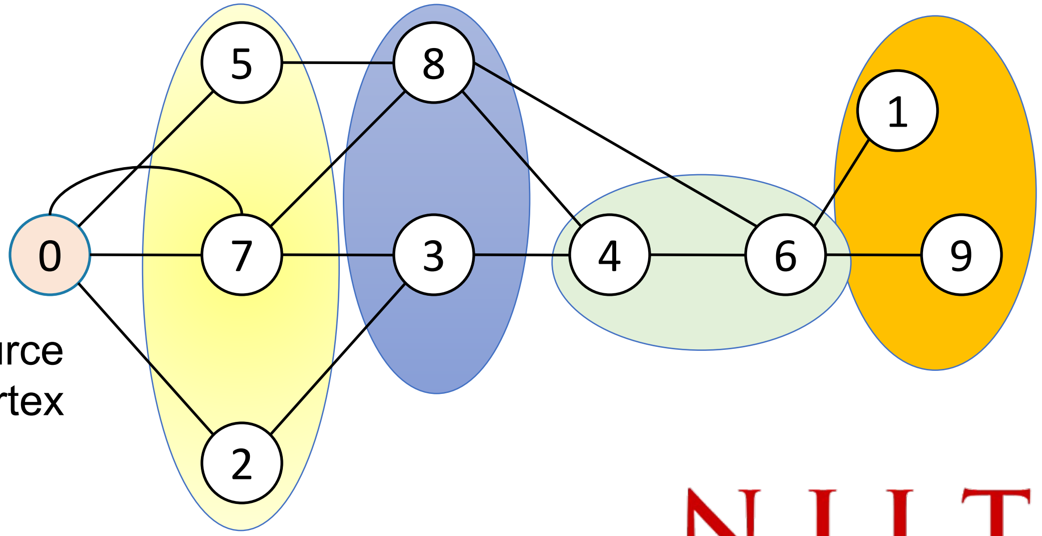
**Output:**




distance from
source vertex

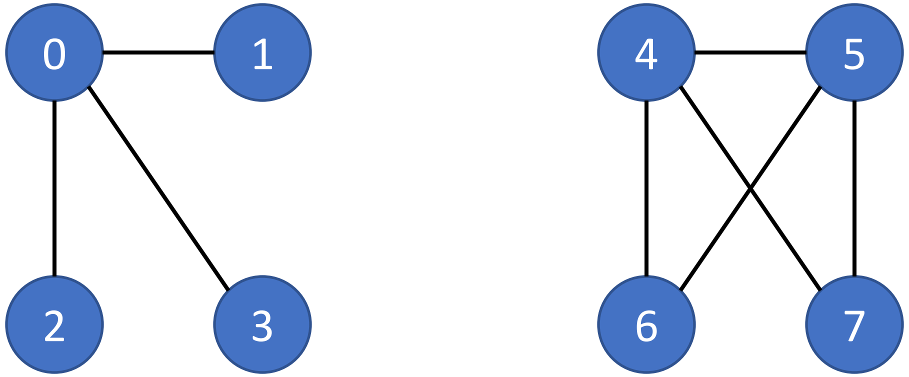



Output: D = [0, 4, 1, 2, 3, 1, 3, 1, 2, 4]

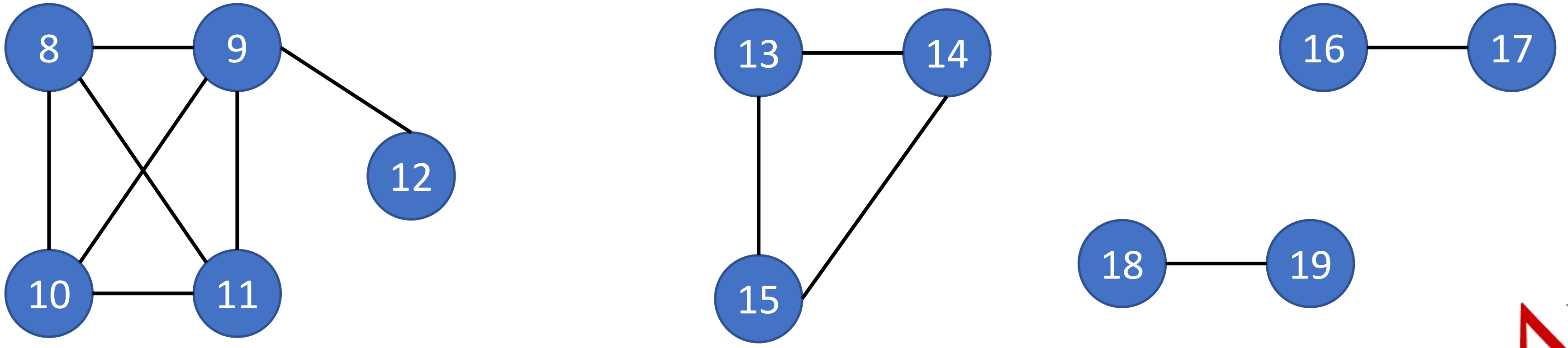



NJIT
New Jersey Institute
of Technology

# Connected Components Example



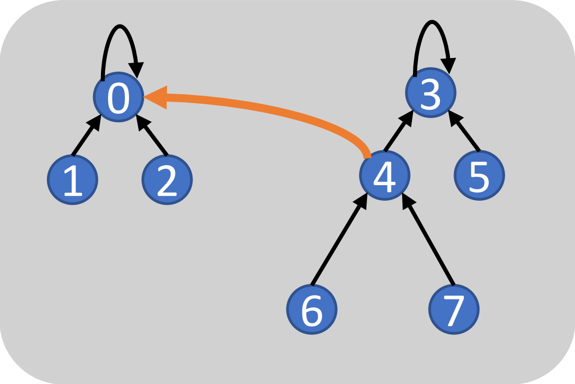

- Connected subgraphs of a graph that is not part of a larger connected subgraph.
- If $u$ is in connected component 1 and $v$ is in connected component 2, there NOT a possible path $u \rightarrow v$.

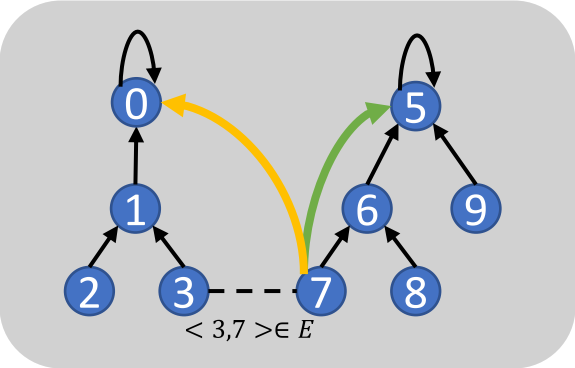


# FastSV Connected Components

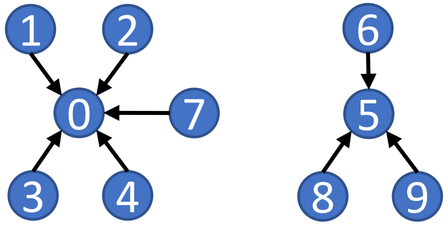
**Algorithm 3.2** Parallel Shiloach-Vishkin connected components.

```
procedure FastSV(G)                                   ▷ A graph G.
    forall u ∈ V do
        L[u] = u
        L_n[u] = u
    end forall
    while L is changing do
        forall e = ⟨u, v⟩ ∈ E do
            L_n[L[u]] = min(L_n[L[u]], L[L[v]])        ▷ stochastic hooking
        end forall
        forall e = ⟨u, v⟩ ∈ E do
            L_n[u] = min(L_n[u], L[L[v]])              ▷ aggressive hooking
        end forall
        forall u in V do
            L_n[u] = min(L_n[u], L[L[u]])              ▷ shortcutting
        end forall
    end while
    return L
end procedure
```




Eventually $L_n$ will make a graph that looks like this:




$L_n =$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 5 | 5 | 0 | 5 | 5 |



NJIT New Jersey Institute of Technology

# Contour Connected Components

Why have a bunch of hooking procedures if we are just propagating labels?

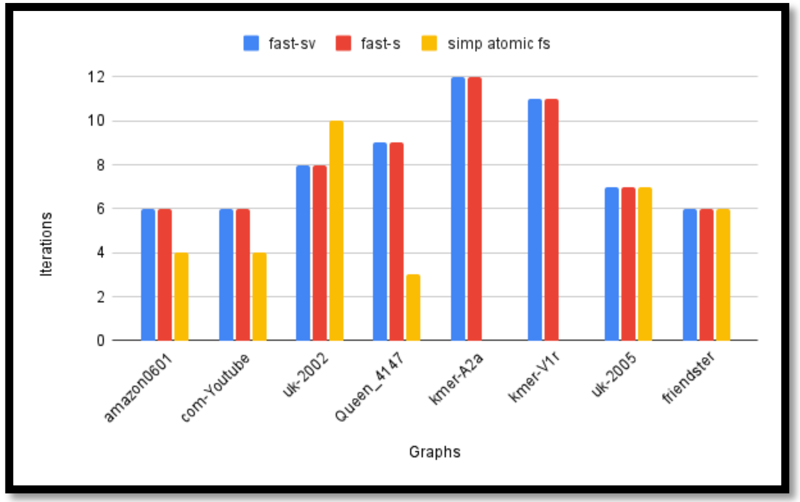
**Takeaway:** Fast-spreading algorithm completes in $O(\log(d_{max}) + 1)$ iterations where $d_{max}$ is the diameter for the largest connected component.

**Algorithm 3.3** Parallel fast-spreading voltage-based connected components.

```
procedure FASTSPREADING(G)                          ▷ A graph G.
    forall i in 0..n − 1 do
        L[i] = i
        L_n[i] = i
    end forall
    while L is changing do
        forall e = ⟨u, v⟩ ∈ E do
            VO²(L_n, L, u, v)
        end forall
        L = L_n
    end while
    return L
end procedure
```

$$VO^2(L_n, L, u, v) : \begin{bmatrix} L_n[u] \\ L_n[v] \\ L_n[L[u]] \\ L_n[L[v]] \end{bmatrix} \xleftarrow{\geq} z^2.$$

$$z^2 = min(L[L[u]], L[L[v]])$$




**RED**=**BLUE** for iterations

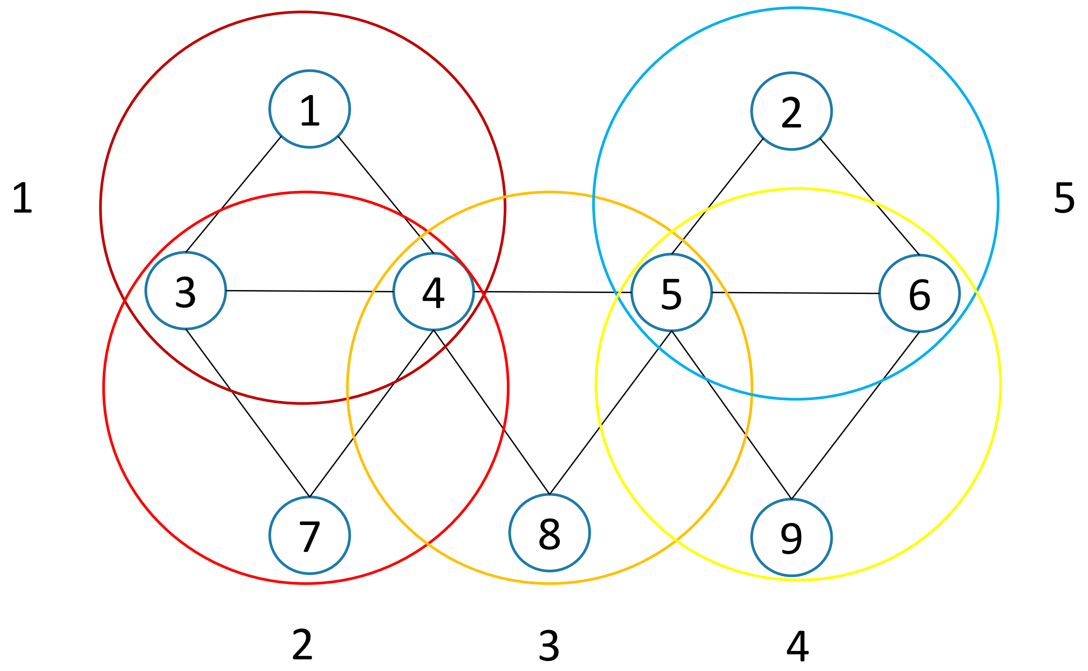
Basically, arriving to same Shiloach-Vishkin result but with less synchronizations.

Preliminary results show same number of iterations as FastSV and faster for some graphs.

NJIT New Jersey Institute of Technology

# Triangle Counting Example

**Input**:




Triangle_count = 5



New Jersey Institute of Technology

# Minimum Search Triangle Counting Algorithm

[Du, Patchett, Alvarado Rodriguez, Li, Bader, 2022]

**Algorithm 3:** *Parallel Minimum Search* based Triangle Counting

**Input:** A graph $G = (V, E)$.

**Output:** An integer value of the number of triangles.

```
coforall loc in Locales do
    forall (edge e = u,v ∈ E) && (e is local) do
        // We assume that |Adj(u)|<|Adj(v)|
        var count:int=0;
        forall w ∈ Adj(u) with (+ reduce Count ) do
            if (|Adj(w)| < |Adj(v)|) then
                if (v ∈ Adj(w)) then
                    count++;
                end
            end
            else
                if (w ∈ Adj(v)) then
                    count++;
                end
            end
        end
    end
end
return count
```

get smallest adjacency list

get closing triangle edge

**If we assume $|Adj_u| < |Adj_v|$, and spawn threads for every $w \in |Adj_u|$, then the running time is:**

Minimum search: $\max_{w \in Adj_u} \log_2(\min(|Adj_w|, |Adj_v|))$

List Intersection: $\log_2(|Adj_v|)$

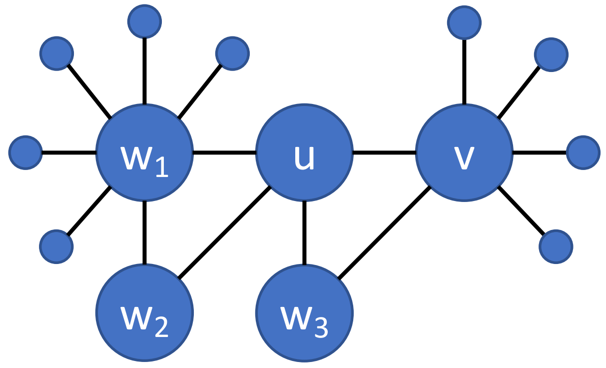


New Jersey Institute of Technology

# Minimum Search Triangle Counting Example

[Du, Patchett, Alvarado Rodriguez, Li, Bader, 2022]

1. Given an edge $(u, v)$ we assume that $|Adj(u)| \leq |Adj(v)|$.
2. Then, for $\forall w \in Adj(u)$ we spawn $|Adj(u)|$ - 1 parallel threads to check if we can form a complete triangle with $(u, v, w)$.
3. If $|Adj(w)| < |Adj(v)|$ we will check if $v \in Adj(w)$, else, we check if $w \in Adj(v)$.




| Adj(x) | Value |
|---|---|
| Adj(u) | 4 |
| Adj(v) | 6 |
| Adj(w1) | 7 |
| Adj(w2) | 2 |
| Adj(w3) | 2 |

Thread $w_1$: search for $w_1$ in $Adj(v)$, no match, kill.

Thread $w_2$: search for $v$ in $Adj(w_2)$, no match, kill.

Thread $w_3$: search for $v$ in $Adj(w_3)$, match! Increment count.

# Minimum Search Triangle Counting Operation Count Comparison

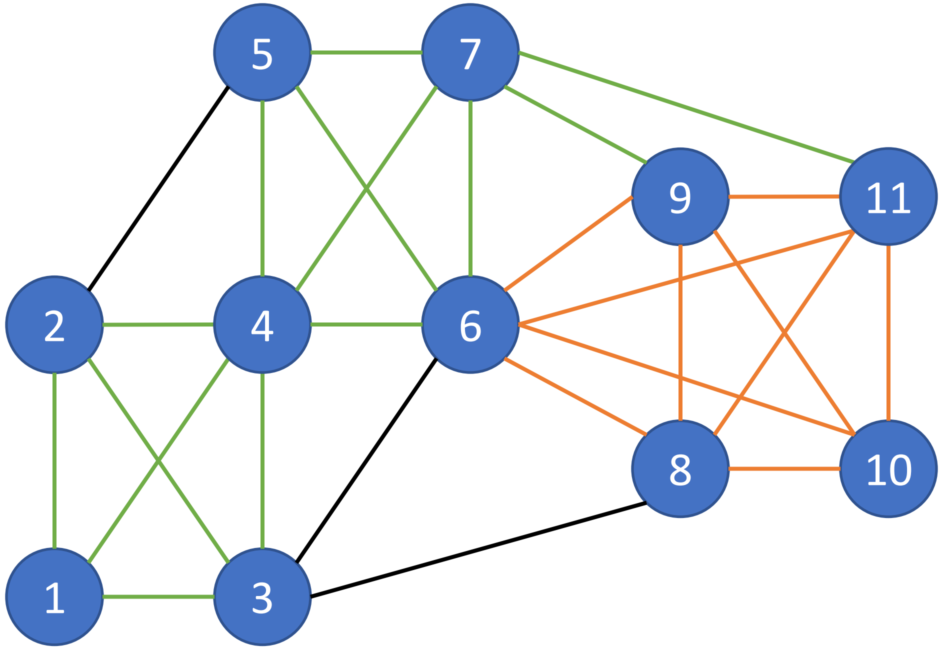
[Du, Patchett, Alvarado Rodriguez, Li, Bader, 2022]

- Assume $|Adj_u| < |Adj_v|$ and we spawn threads for every $w \in |Adj_u|$
  - Minimum search: $\max_{w \in Adj_u} \log_2(\min(|Adj_w|, |Adj_v|))$
  - List Intersection: $\log_2(|Adj_v|)$
- Say we have the following information for our vertices:
  - $|Adj_u| = 4$ and $|Adj_v| = 1024$
  - For every $w$ in $Adj_u$, $|Adj_w| \leq 8$
- **List intersection:** 4 threads amounting to $\lceil \log_2 1024 \rceil = 10$ operations each.
- **Minimum search:** 4 threads amounting to $\lceil \log_2 8 \rceil = 3$ operations each.



New Jersey Institute of Technology

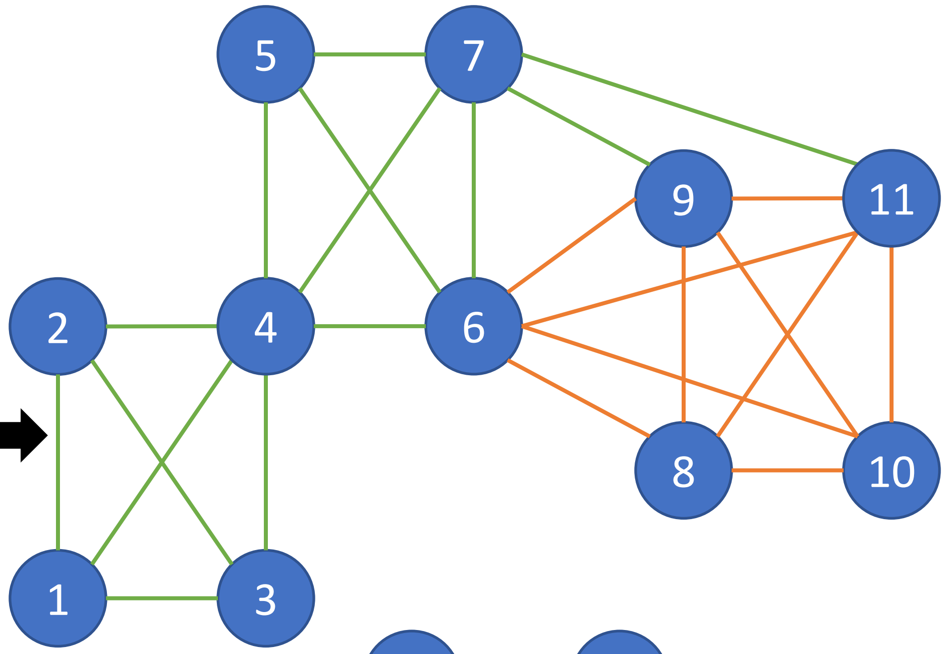
# Truss Analytics Example



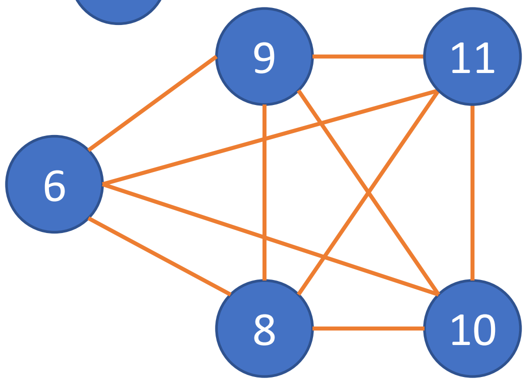

**3-truss**
**4-truss**
**5-truss** = max-truss

coloring = truss decomposition

k=4 truss




k=5 = max-truss




truss decomposition

[3, 3, 3, ............., 4,5,5,5,5,5,5,5,5,5,5]
edge id with trussness

Every edge is part of at least k-2 triangles.

# K-truss High-Performing Algorithm [Du, Patchett, Alvarado Rodriguez, Li, Bader, 2022]

**Algorithm 4:** Optimized *k-Truss* Parallel Algorithm

```
OptKTruss(G, k)
var JustDelEBag= new DistBag(int,Locales);
AtoSup ← 0 and EDel ← -1
TriangleCounting(G, EDel, AtoSup)
coforall loc in Locales do
    forall (e ∈ E) && (e is local) do
        if (EDel[e]==-1) && (AtoSup[e].read() < k-2)
         then
            EDel[e]=1-k
            JustDelEBag.add(e);
        end
    end
end
while (JustDelEBag.getSize() > 0 ) do
    SupportUpdate(G, EDel, JustDelEBag, AtoSup)
    coforall loc in Locales do
        forall (e ∈ JustDelEBag) && (e is local) do
            if (EDel[e]==1-k) then
                EDel[e]=k-1
            end
        end
        JustDelEBag.clear();
        forall (e ∈ E) && (e is local) do
            if (EDel[e]==-1) &&
             (AtoSup[e].read() < k-2) then
                EDel[e]=1-k
                JustDelEBag.add(e);
            end
        end
    end
end
return EDel
```

distribution and edge parallelism

generate new support
*only with undeleted edges*

keep removing edges with support less than k-2 until all proper supports have been updated

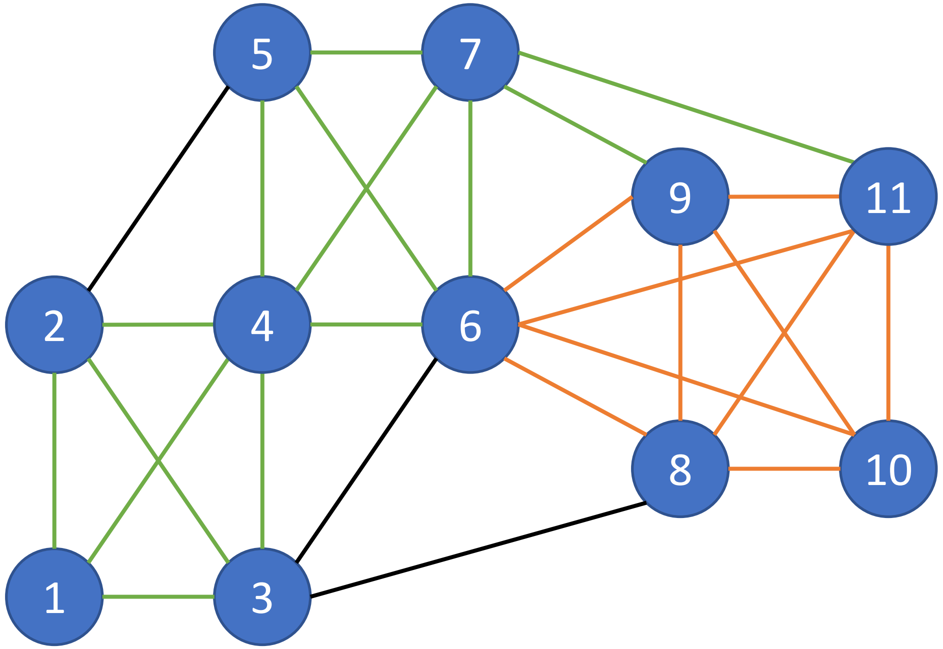
**Algorithm 2:** *Minimum Search* based Support Updating

```
SupportUpdate(G, EDel, JustDelEBag, AtoSup)
coforall loc in Locales do
    forall (e1 = ⟨u, v⟩ ∈ JustDelEBag) && (e is local)
     do
        /* We assume that |Adj(u)| < |Adj(v)|    */
        forall (e2 = ⟨u, w⟩, w ∈ Adj(u) − {v}) &&
         (EDel[e2] < 0) do
            Search e3 = ⟨v, w⟩ or e3 = ⟨w, v⟩;
            if (e3 exists) then
                if (e2 and e3 are undelted) then
                    AtoSup[e2].sub(1);
                    AtoSup[e3].sub(1);
                end
                else
                    if (e2 is undeleted) && (e1 < e3) then
                        AtoSup[e2].sub(1);
                    end
                    if (e3 is undeleted) && (e1 < e2) then
                        AtoSup[e3].sub(1);
                    end
                end
            end
        end
    end
end
```

**fun fact:** beginning iteration (+ reduce AtoSup))/3 = tricount

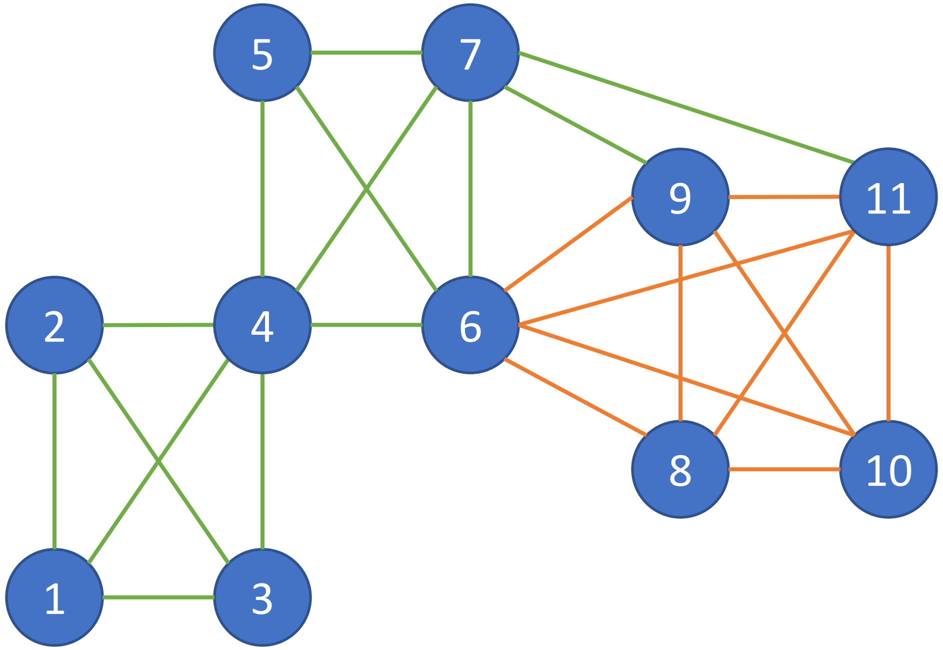
Visual K-Truss Example [Du, Patchett, Alvarado Rodriguez, Li, Bader, 2022]

Example: Find k=5 truss



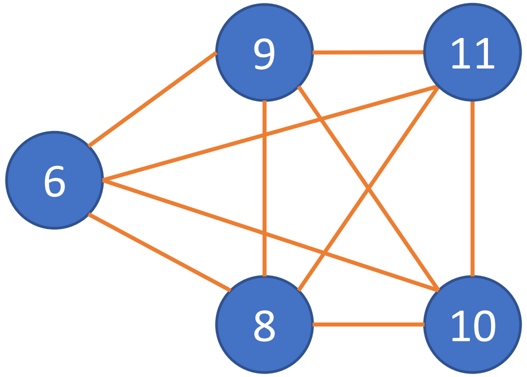

Delete black edges.

Update supports.




Delete green edges.

Update supports.



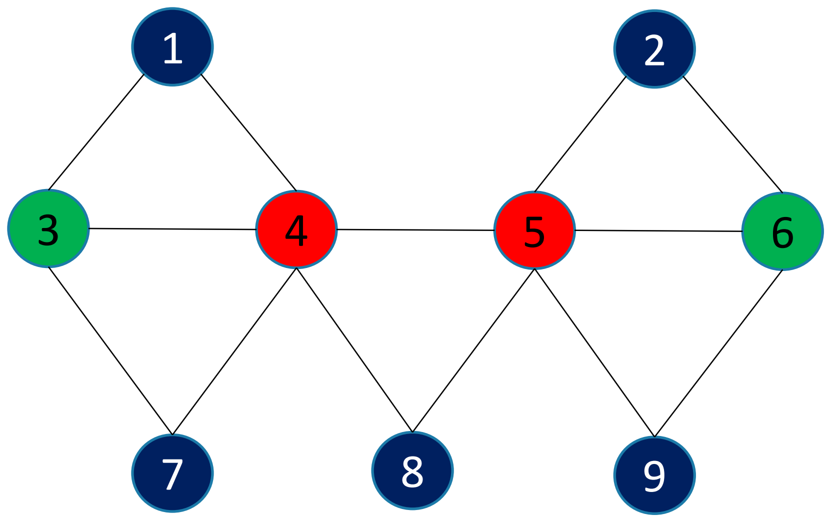

Output



New Jersey Institute of Technology

# Triangle Centrality Example

**Input**:




$$TC(v) = \frac{\frac{1}{3}\sum_{u \in N^{+}_{\triangle}(v)} \triangle(u) + \sum_{w \in \{N(v) \setminus N_{\triangle}(v)\}} \triangle(w)}{\triangle(G)}.$$

[Burkhardt, 2021]

**Output:** D = [**0.4**, **0.4**, **0.47**, **0.73**, **0.73**, **0.47**, **0.4**, **0.4**, **0.4**]



NJIT New Jersey Institute of Technology

# What kind(s) of questions can each algorithm answer?

- **Breadth-first search**
  - **How far is vertex u from a vertex v? Is a vertex v reachable from a vertex u?**
  - **What is the diameter of a graph?**
  - **How are the nodes distributed amongst depths based off a source vertex?**
- **Connected components**
  - **What is the largest connected component in a graph?**
  - **What is the distribution of the sizes of all the connected components?**
  - **Is the graph connected?**
- **Triangle counting**
  - **Most obvious: how many triangles (3-cliques) in the graph?**
  - **How many triangles are adjacent to a vertex u?**
- **Truss analytics**
  - **Is there a subgraph where every edge belongs to [2,3,+] triangles?**
  - **What is the maximum number of triangles every edge belongs to?**
- **Triangle centrality**
  - **How important is a vertex u dependent on how many triangles are around it?**
  - **How does this match with other metrics such as betweenness centrality?**

# Jupyter Notebook Time ☺



New Jersey Institute of Technology

Thank You ☺
Questions?



New Jersey Institute of Technology