Structural Cohesion & Worldly Nodes
Solving for large-network connectivities
(part II)

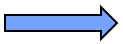
Doug White
IMBS at UC Irvine & Santa Fe Institute
and Bob Sinkovits at SDSC

Finding k-cohesive subgroups

- Construct pair-wise cohesion matrix M (the hard part!)
- For each value of k
  - If $m_{ij} \geq k$, then $m_{ij} \leftarrow 1$; otherwise $m_{ij} \leftarrow 0$
  - Identify cliques in resulting adjacency matrix

0	2	1	3	3
2	0	1	1	1
1	1	0	1	1
3	1	1	0	3
3	1	1	3	0

Cohesion matrix

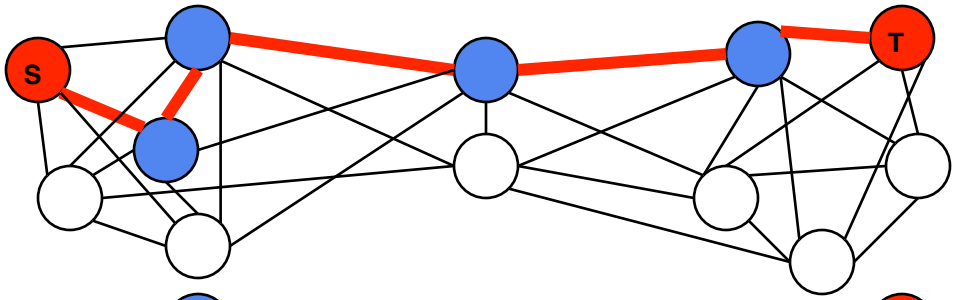


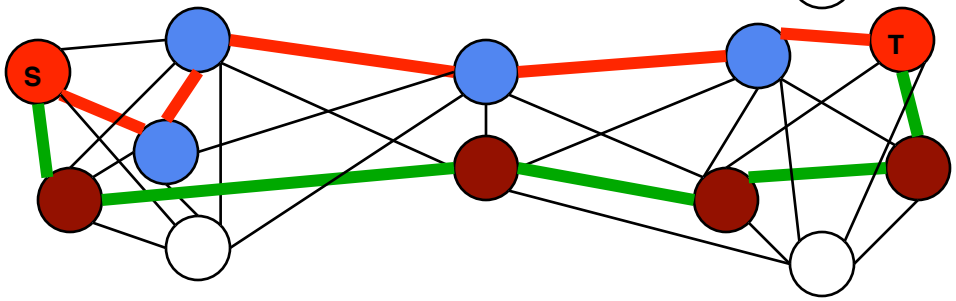
0	0	0	1	1
0	0	0	0	0
0	0	0	0	0
1	0	0	0	1
1	0	0	1	0

Adjacency matrix

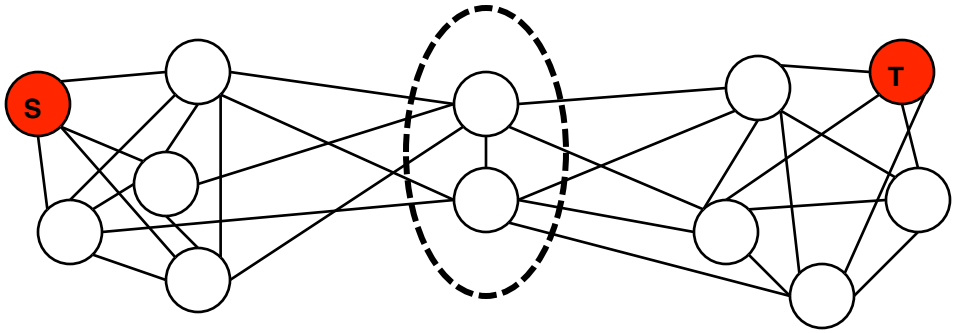
Vertices (1,4,5) form 3-cohesive subgroup

Menger's Theorem Refresher





Number of vertex disjoint paths (no two simultaneous paths share a vertex)



Minimum number of vertices that need to be removed so that source and target are no longer connected

Co-author data set

Data set:

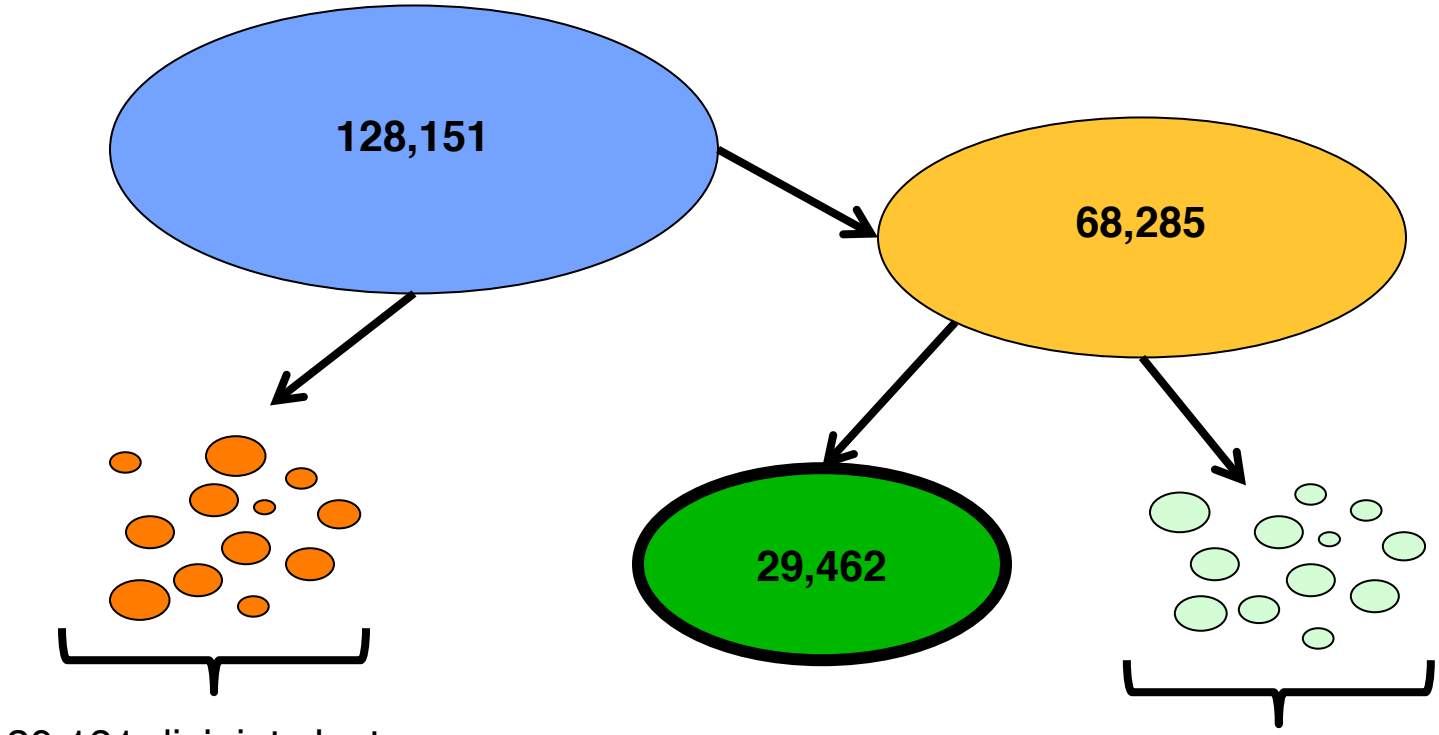
- After we developed parallel algorithm, decided to tackle co-author data set sociology journals 1963-99 (vertices are authors, edges connect co-authors)
- Told that this contained 13,000 vertices – big, but we weren't intimidated
- Turned out to be 130,000 vertices. At least 1000x harder than a problem that was already considered to be pushing the limits!!!

Strategy:

- Reduce to a simpler problem
- Be clever about filling in elements of cohesion matrix
- Once we're out of steam, resort to supercomputing to finish off problem

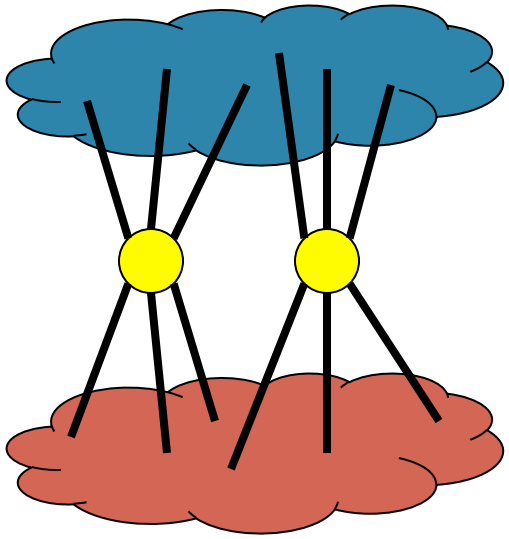
Largest biconnected component

Since we're interested in cohesion, focus on connected clusters. Within a connected cluster everyone is trivially 1-connected, focus on the biconnected components.



Filling in the 2s in the cohesion matrix

Starting with a biconnected component, we know that every pair of vertices is at least 2-connected. If we can find all of the 2-vertex separators, then we can find **ALL** pairs of vertices that are joined by exactly two vertex disjoint paths

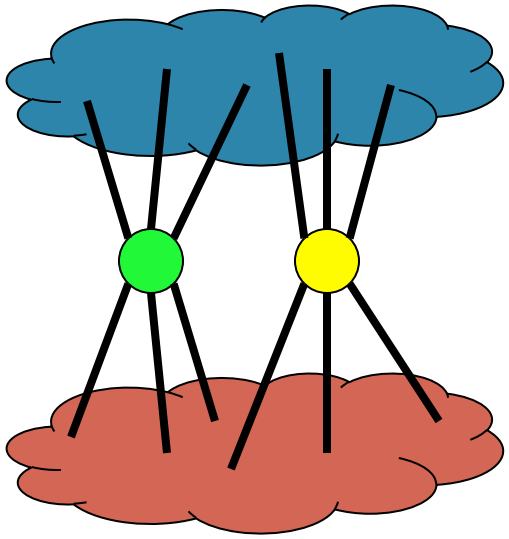


The pair of yellow vertices is a 2-vertex separator. Since the entire graph is biconnected, any pair of vertices with one member in D_1 and the other in D_2 is exactly 2-connected

$$N_{VDP}(x \in D_1, y \in D_2) = 2$$

Finding the 2-vertex separators

Testing a set of vertices for the property of being a vertex separator is fairly time consuming and explicitly testing every pair can take a long time. Fortunately, there are very fast algorithms for finding articulation points (1-vertex separators)



D_1

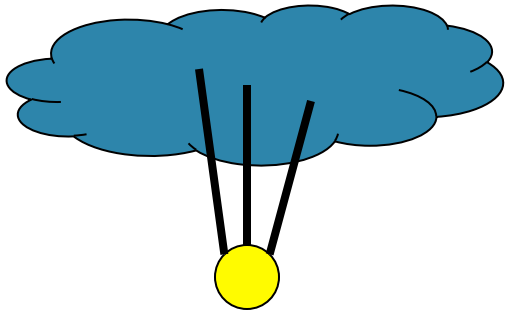
Removing green vertex from graph makes the yellow vertex an articulation point (1-vertex sep)

D_2

$$
\begin{aligned}
&Loop\ over\ v_1 \in V \\
&\quad G' \leftarrow G - v_1 \\
&\quad v_{art} \leftarrow ArtPts(G') \\
&\quad Loop\ over\ v_2 \in v_{art} \\
&\qquad S_2 \leftarrow S_2 + \{v_1, v_2\}
\end{aligned}
$$

Filling in some of the 3s

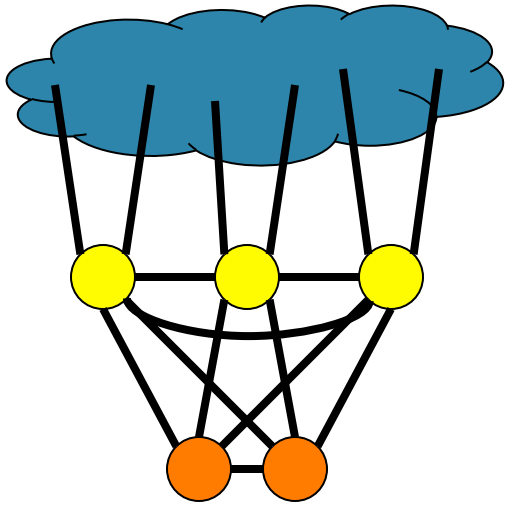
Now that we've found all of the exactly 2-connected pairs, every remaining pair is at least 3-connected. Any pair of vertices where one or both vertices is of degree 3 will be exactly 3-connected.



- Yellow vertex is of degree 3
- Max N_{VDP} between pair of vertices is the minimum of the target/source vertex degrees
- Therefore, unless N_{VDP} previously set to 2, all pairs involving the yellow vertex will have exactly 3 VDPs

Filling in more of the 3s

Finding all of the 3-vertex separators is very time consuming, but we can make an educated guess to find many of the 3-vertex separators. Test the neighbors of high-degree vertices (d > 3)



The orange vertices are both of degree 4. Testing all triplets of neighbors will identify the yellow vertices as a 3-vertex separator. These "isolated" vertices will be 3-connected to all vertices outside clique unless previously determined to be 2-connected

What was the payoff and how long did this take?

Using procedures described earlier, found:

- 329 million 2-connected pairs (76% of all matrix elements)
- 67 million 3-connected pairs (15% of all matrix elements)
- 396 million pairs total (91% of all matrix elements)

The computational expense was

- 3 minutes to find all 2-vertex separators (16-way parallel, one node)
- 10 minutes to fill in the 2s (serial)
- 50 minutes to fill in 3s (serial)

Now it's time to break out the sledgehammer

- Can probably be clever and find more or all of the exactly 3-connected pairs, but (computationally) may not be worth the effort.
- From this point, we simply call the vertex disjoint paths function for every remaining element of the cohesion matrix.

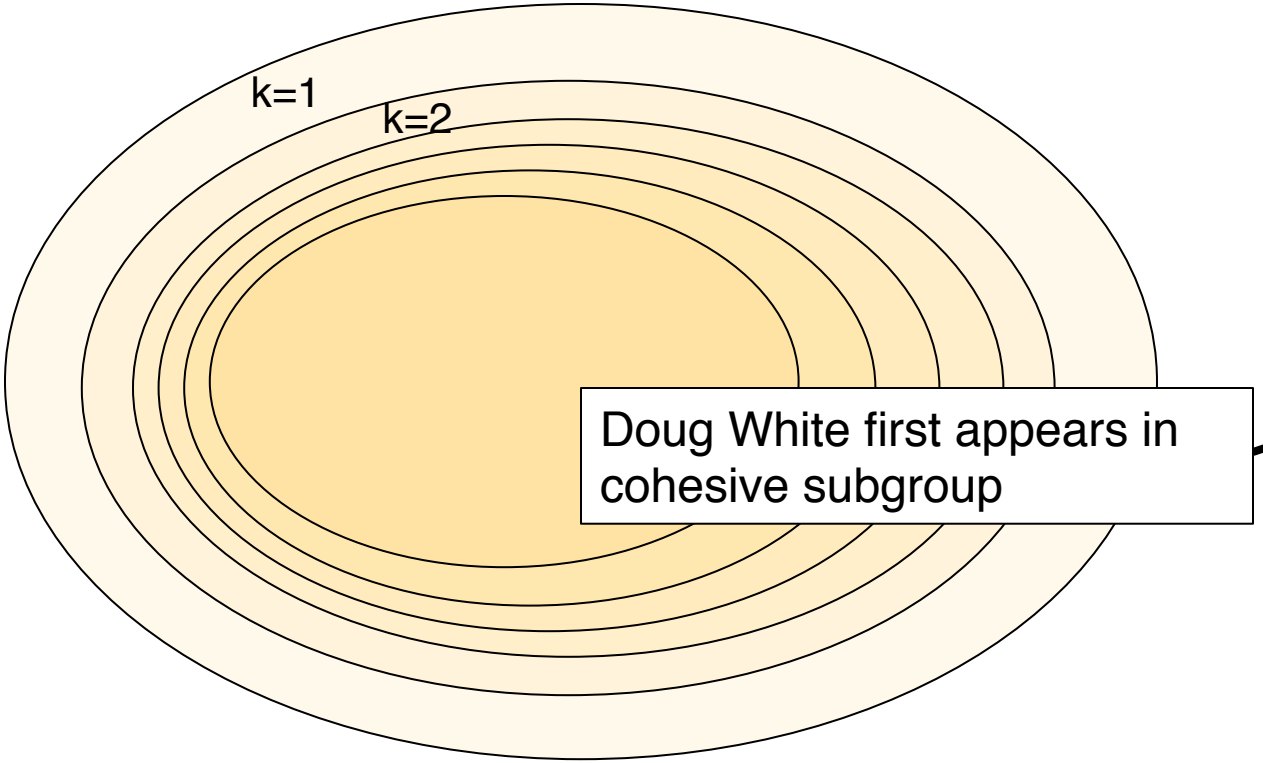
```
for i in V(G)
  for j < i in V(G)
    if m[i,j] not previously calculated
      m[i,j] ← vertex_disjoint_paths(G,i,j)
```

Fortunately we can do this in parallel

- Every pair of vertex disjoint paths calculations is independent. Can simply divide up the work across compute cores.
- Somewhat complicated by the fact that R is single threaded. What looks like a perfect opportunity for thread-level parallelism actually requires spawning new processes and replicating data structures.
- Broke up problem into 9 independent 16-core jobs. Parallelization within job done using mclapply function from R multicore package
- Wall time for this step ~ 6 hours; 7 hours for entire workflow
- Partial results from the 9 jobs combined after all jobs complete
- Once we have the cohesion matrix, 99% of the effort is done

What did we find?

Nested k-cohesive subgroups, dominated by a single large subgroup for 1 ≤ k ≤ 20. Cohesive subgroups found up to k=56



k	size
1	68285
2	29467
3	14513
4	8118
5	5068
6	3275
7	2295
8	1691
9	1235
10	920
11	676
12	529
…	…
56	2

Summary (computations)

- Didn't take much of a performance hit using R since most of the heavy computation was done in the R igraph library (written in C)
- Convenient to work in R – things that should be easy (reading graph, writing results, etc.) were easy. Could prototype using GUI on laptop.
- BUT … the parallelization was a little awkward. An R-equivalent to OpenMP pragmas would have been nice.
- AND… getting close to object size limit (2^{31} elements). Won't be able to do larger than ~46000 vertices using standard R
- Considering redeveloping in C to get around R limitations. Can still rely on the igraph library
- Combination of finesse and brute force was ideal
- May have some room left for clever ideas

Summary (scientific impact)

- No one had ever identified the cohesive subgroups in a graph anywhere close to this size before. Collaborator had once told us this was an intractable problem.
- Won't be able to apply these technique to enormous graphs (internet, Facebook, Twitter), but now feasible for other social science problems and biological networks containing O(10K) genes, proteins, etc.