Computing Strongly Connected Components in Modern Architectures

Shared-Memory Implementation and Testing

George M. Slota[1,2] Sivasankaran Rajamanickam[1]
Erik G. Boman[1]

[1]Sandia National Laboratories
[2]Pennsylvania State University

8 July 2013

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
This work was supported by the DOE Office of Science through the FASTMath SciDAC Institute.

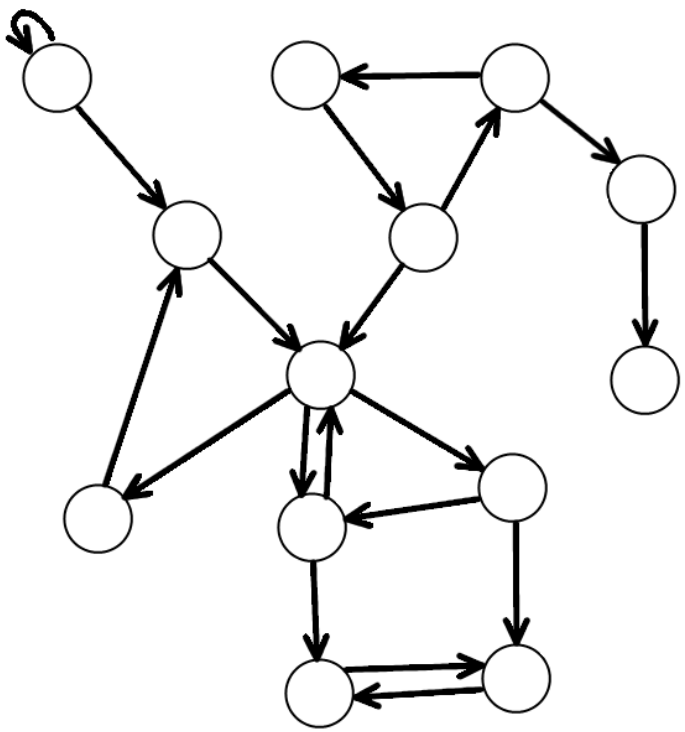
Overview

- Previous parallel strongly connected component (SCC) algorithms
- Current implementation and improvements
- Performance results
- Conclusions and Future work

Previous Related Algorithms

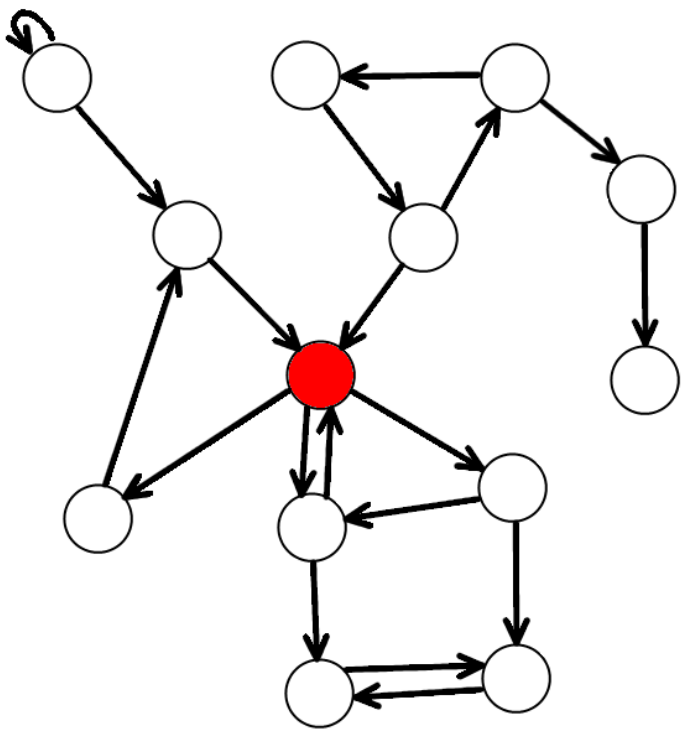
- Forward-Backward (FW-BW) [1]
- Trimming [2]
- Coloring [3]
- Others [4, 5]

Previous Algorithms
Forward-Backward (FW-BW)



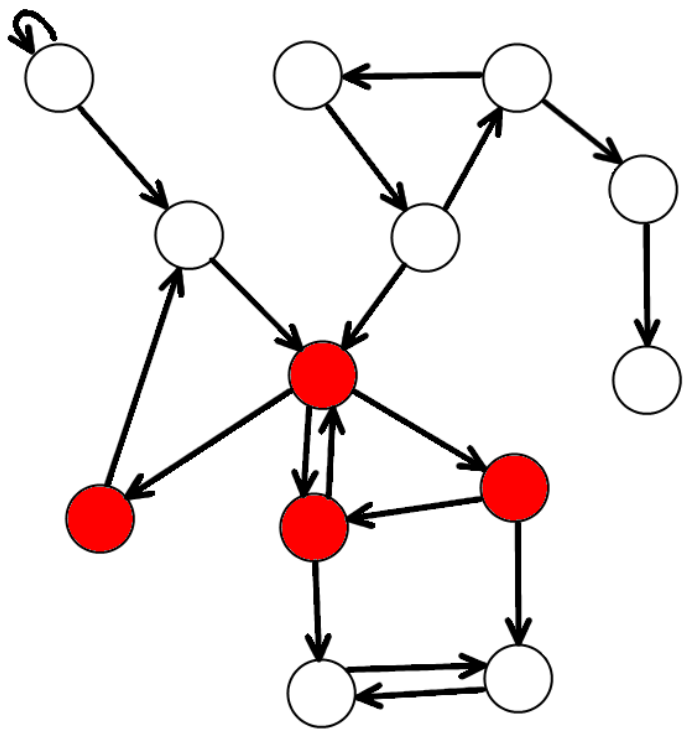
Previous Algorithms
Forward-Backward (FW-BW)

- Select pivot



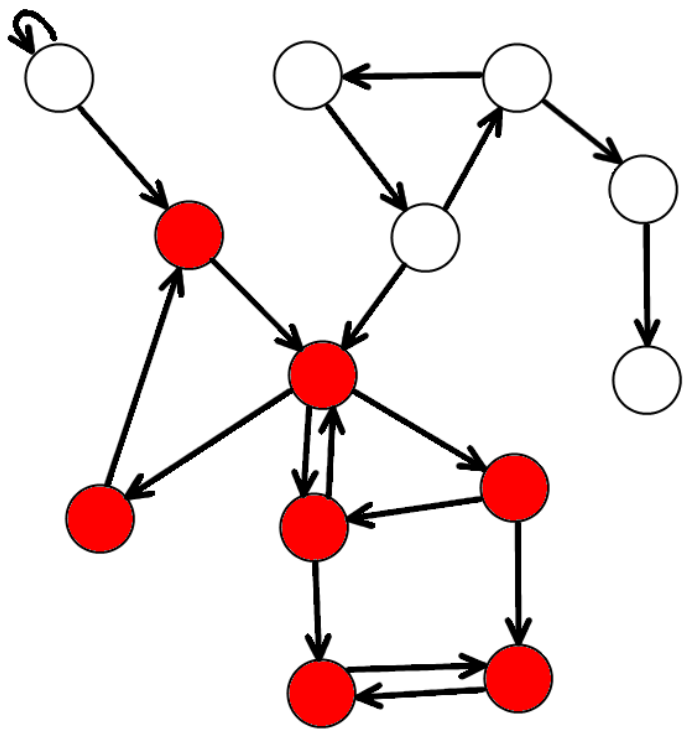
Previous Algorithms
Forward-Backward (FW-BW)

- Select pivot
- Find all **descendant** vertices that can be reached by pivot (D)



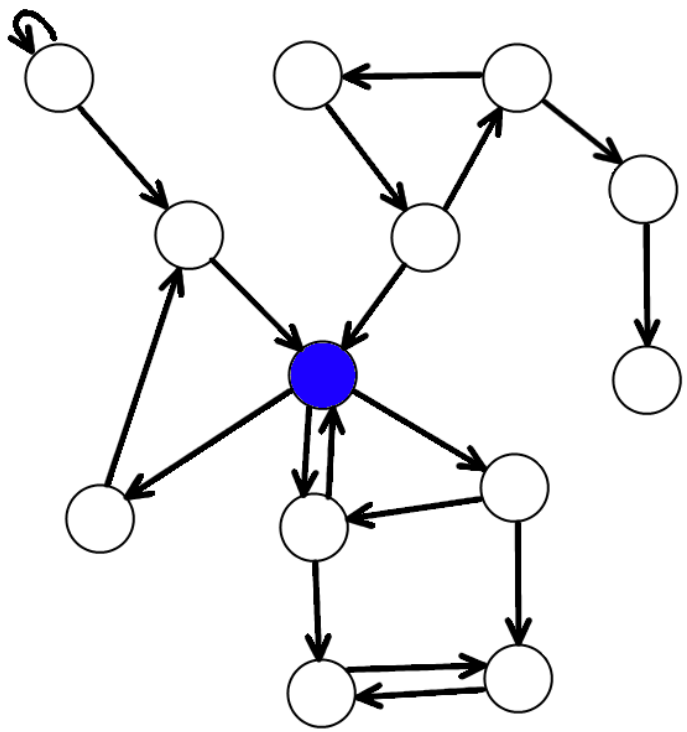
Previous Algorithms
Forward-Backward (FW-BW)

- Select pivot
- Find all **descendant** vertices that can be reached by pivot (D)



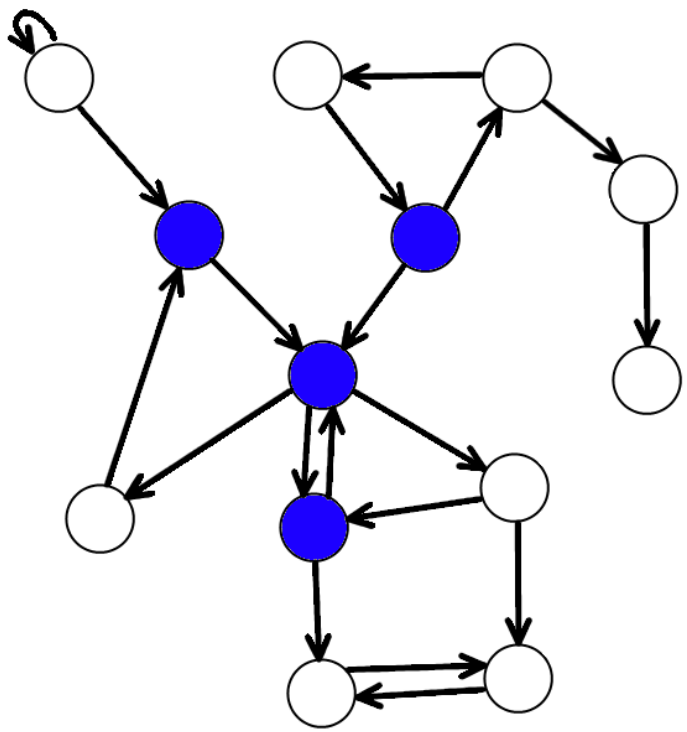
Previous Algorithms
Forward-Backward (FW-BW)

- Select pivot
- Find all **descendant** vertices that can be reached by pivot (D)
- Find all **predecessor** vertices that can reach pivot (P)



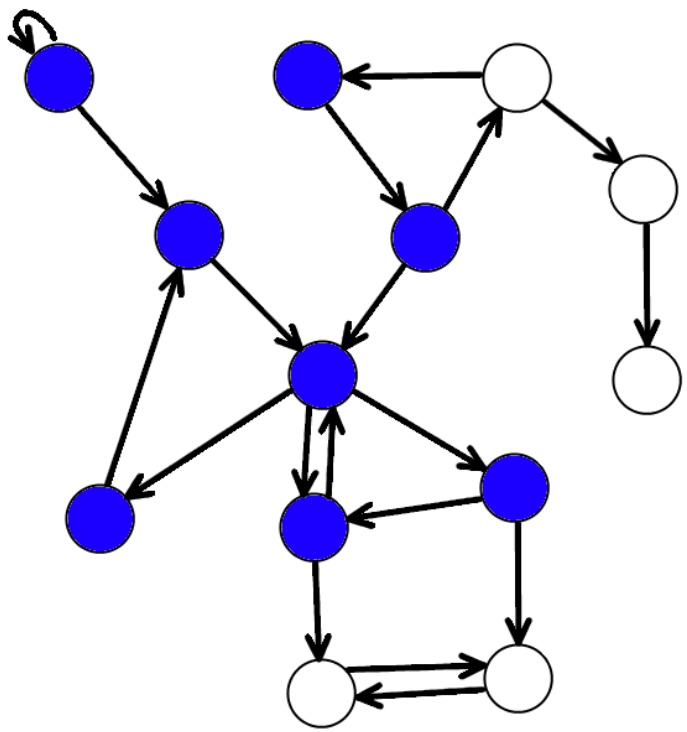
Previous Algorithms
Forward-Backward (FW-BW)

- Select pivot
- Find all **descendant** vertices that can be reached by pivot (D)
- Find all **predecessor** vertices that can reach pivot (P)



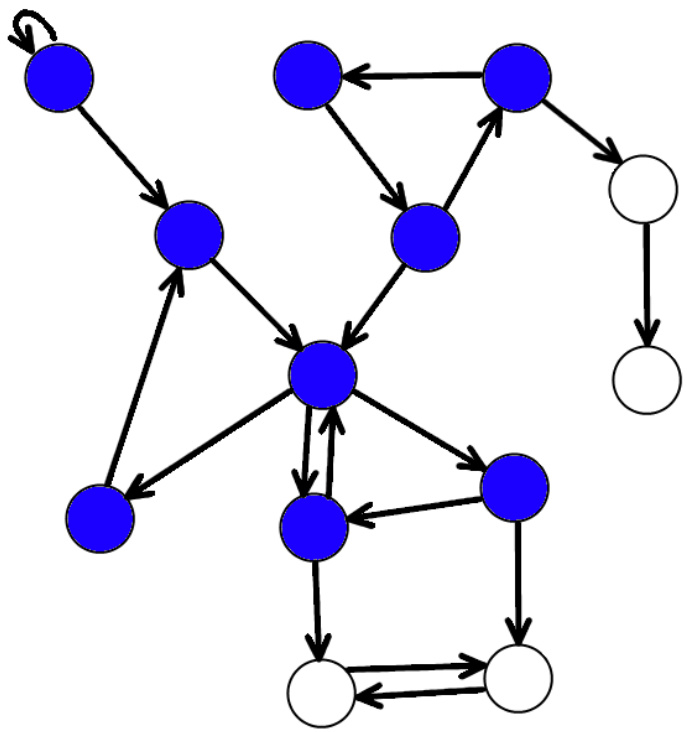
Previous Algorithms
Forward-Backward (FW-BW)

- Select pivot
- Find all **descendant** vertices that can be reached by pivot (D)
- Find all **predecessor** vertices that can reach pivot (P)



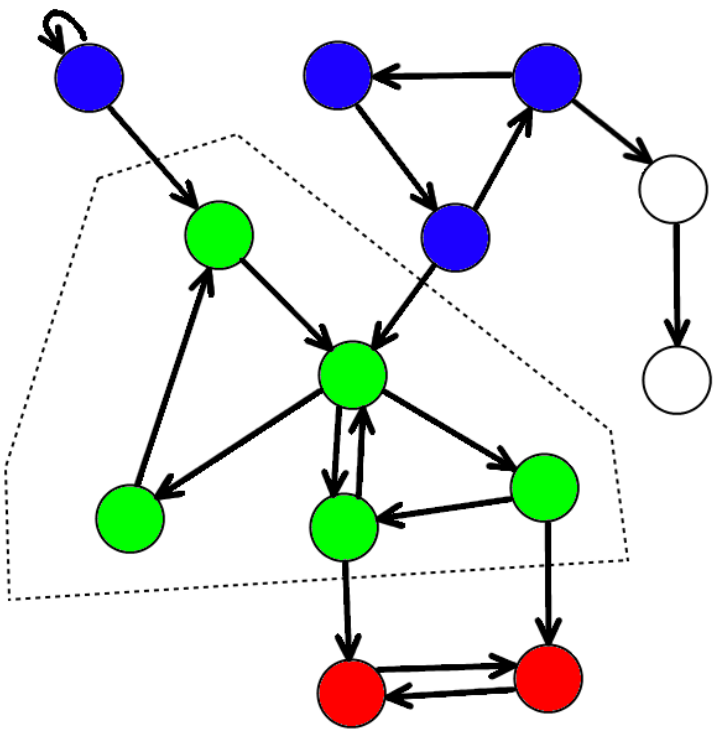
Previous Algorithms
Forward-Backward (FW-BW)

- Select pivot
- Find all **descendant** vertices that can be reached by pivot (D)
- Find all **predecessor** vertices that can reach pivot (P)



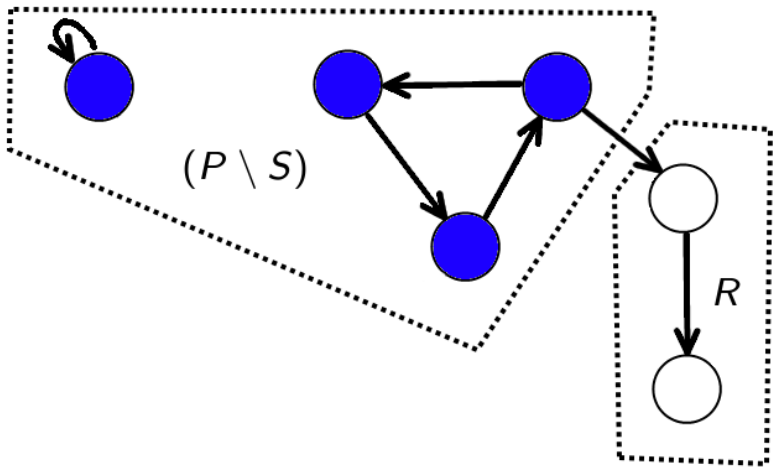
Previous Algorithms
Forward-Backward (FW-BW)

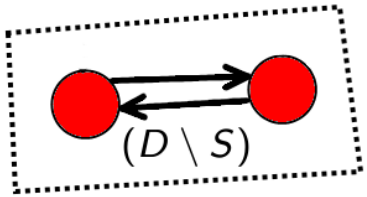
- Select pivot
- Find all **descendant** vertices that can be reached by pivot (D)
- Find all **predecessor** vertices that can reach pivot (P)
- Intersection of those two sets is an SCC ($S = P \cap D$)



Previous Algorithms
Forward-Backward (FW-BW)

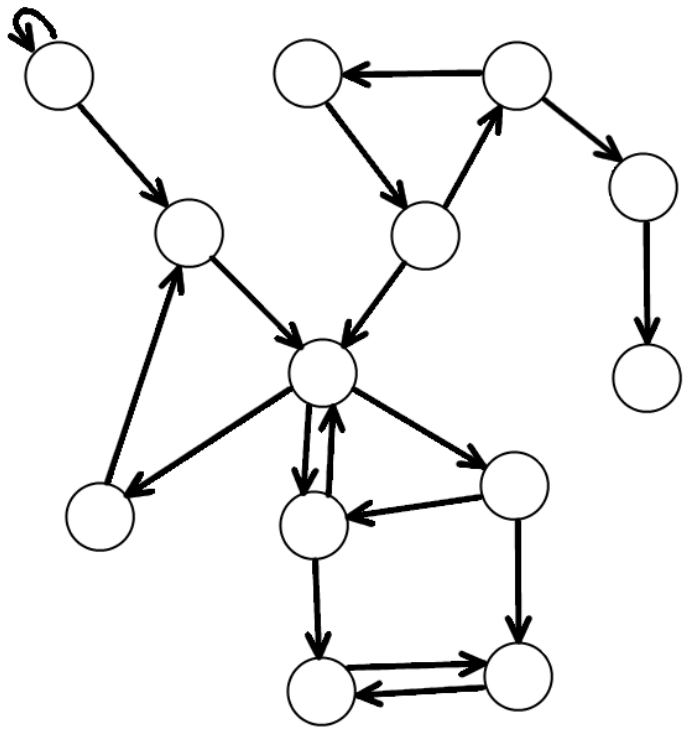
- Select pivot
- Find all **descendant** vertices that can be reached by pivot (D)
- Find all **predecessor** vertices that can reach pivot (P)
- Intersection of those two sets is an SCC $(S = P \cap D)$
- Now have three distinct sets leftover $(D \setminus S), (P \setminus S)$, and **remainder** (R)





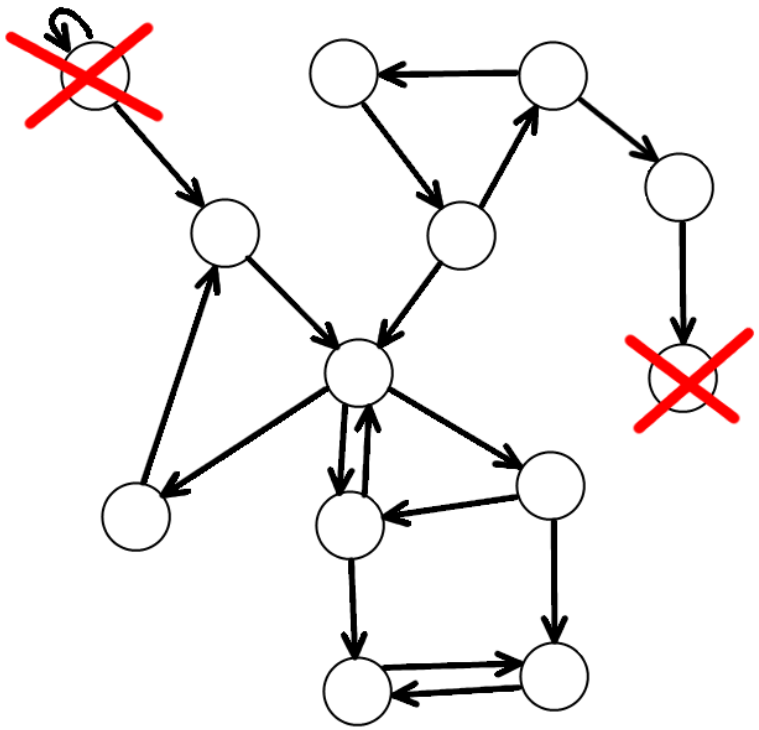
Previous Algorithms
Trimming

- Used to find trivial SCCs



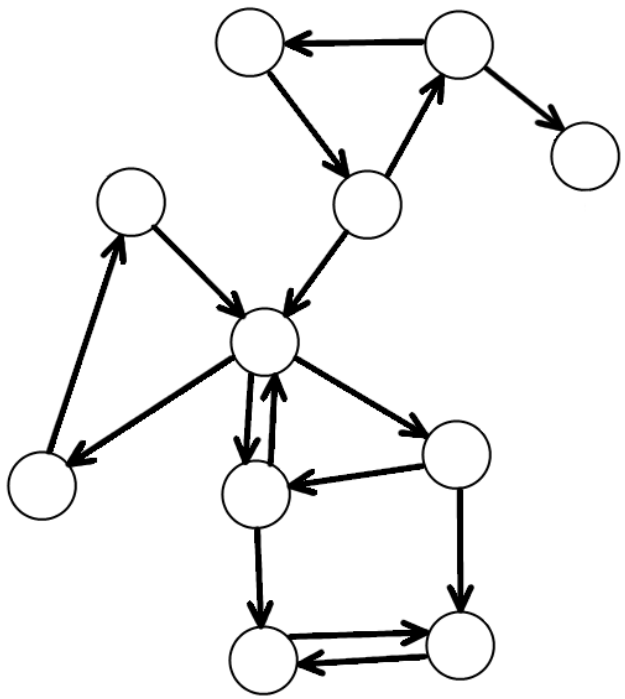
Previous Algorithms
Trimming

- Used to find trivial SCCs
- Detect and prune all vertices that have an in/out degree of 0 or an in/out degree of 1 with a self loop (simple trimming)



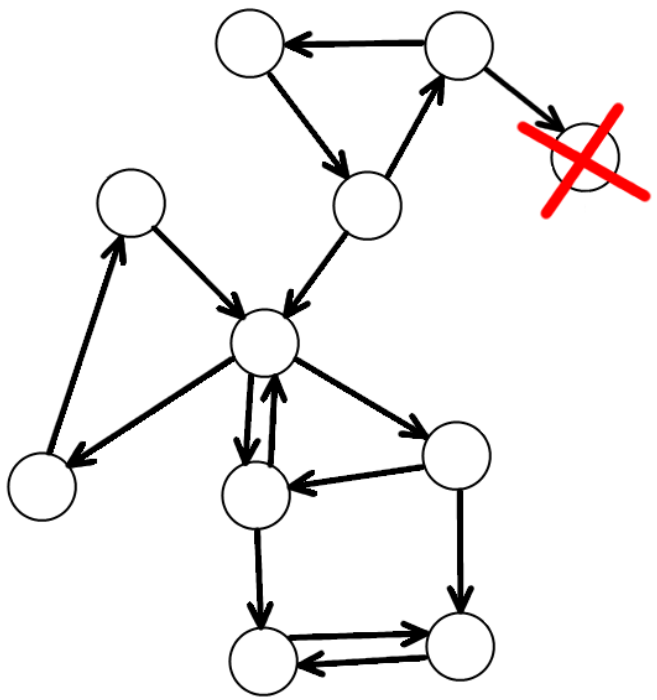
Previous Algorithms
Trimming

- Used to find trivial SCCs
- Detect and prune all vertices that have an in/out degree of 0 or an in/out degree of 1 with a self loop (simple trimming)



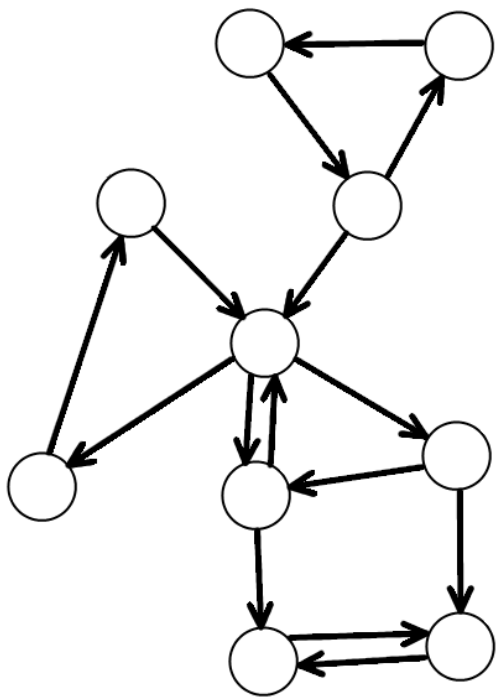
Previous Algorithms
Trimming

- Used to find trivial SCCs
- Detect and prune all vertices that have an in/out degree of 0 or an in/out degree of 1 with a self loop (simple trimming)
- Repeat iteratively until no more vertices can be removed (complete trimming)



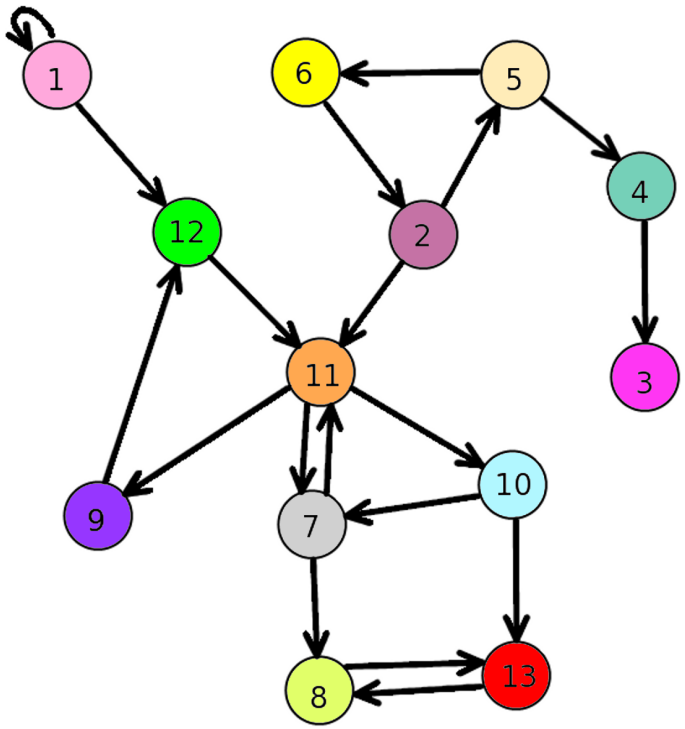
Previous Algorithms
Trimming

- Used to find trivial SCCs
- Detect and prune all vertices that have an in/out degree of 0 or an in/out degree of 1 with a self loop (simple trimming)
- Repeat iteratively until no more vertices can be removed (complete trimming)



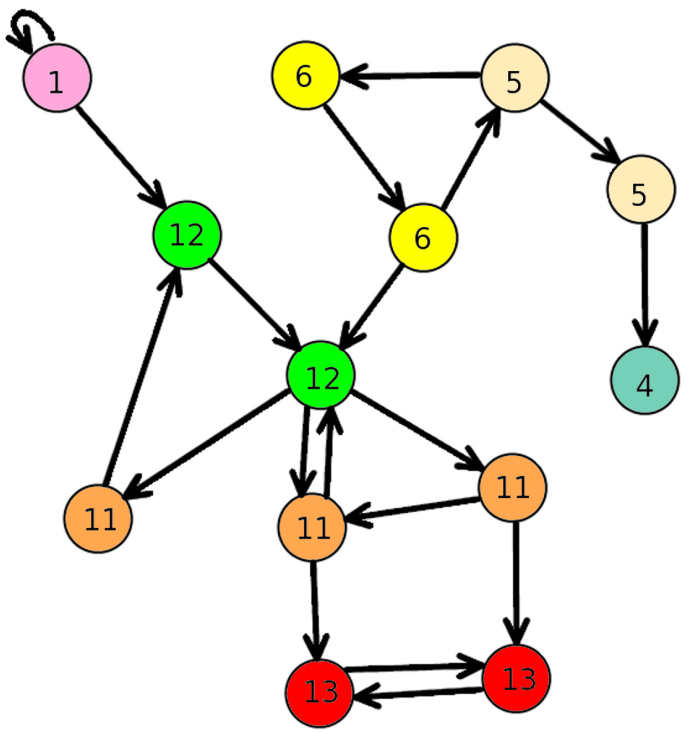
Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*



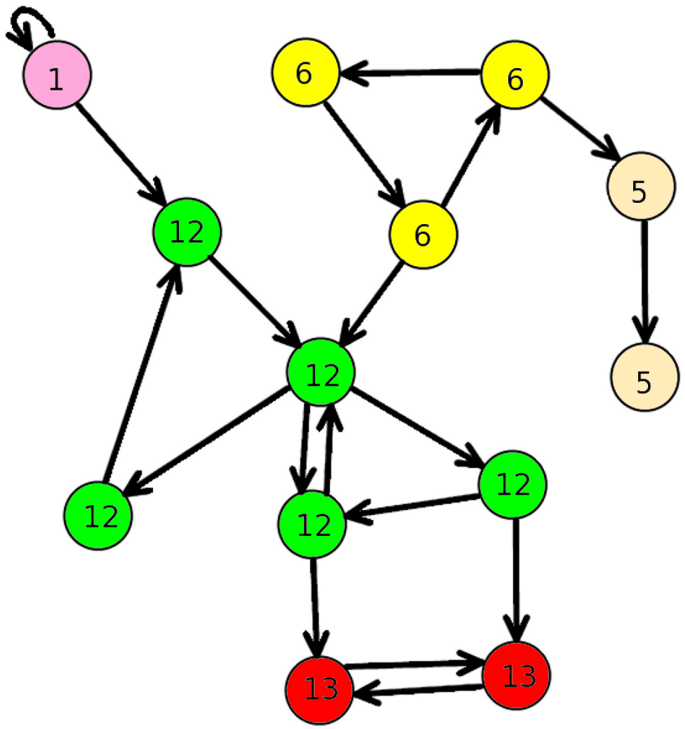
Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets



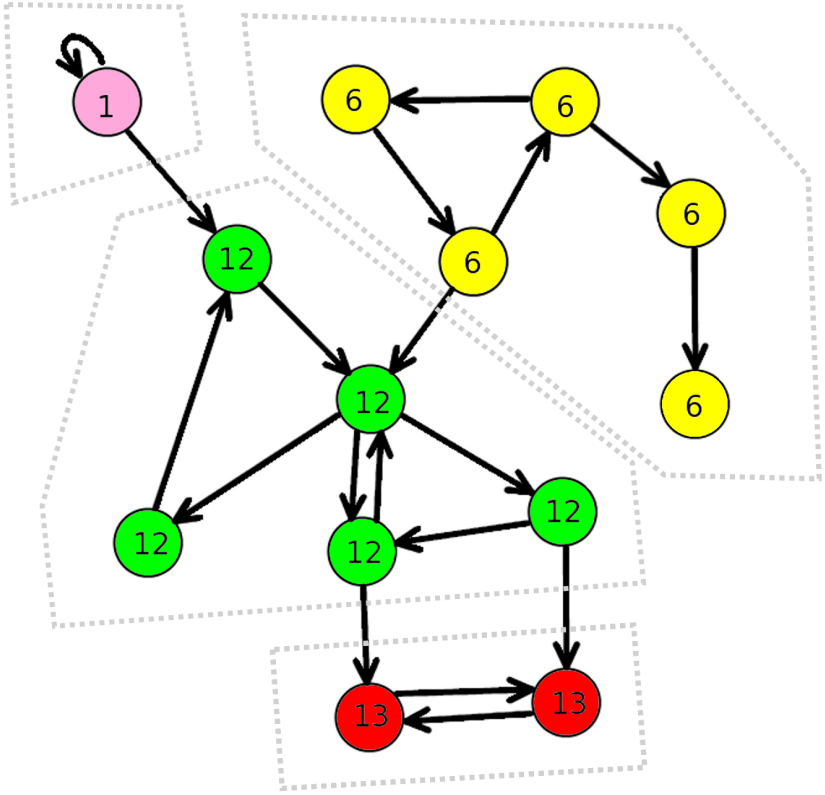
Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets



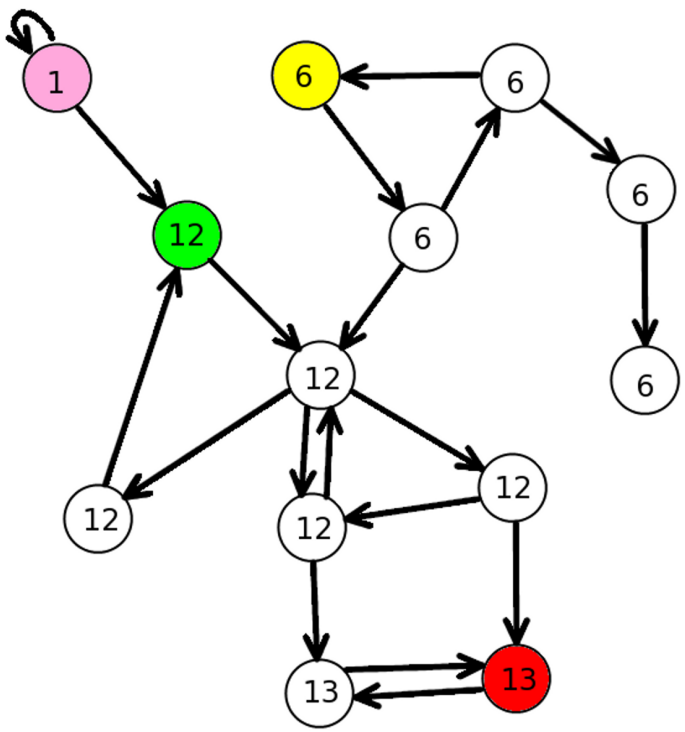
Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets



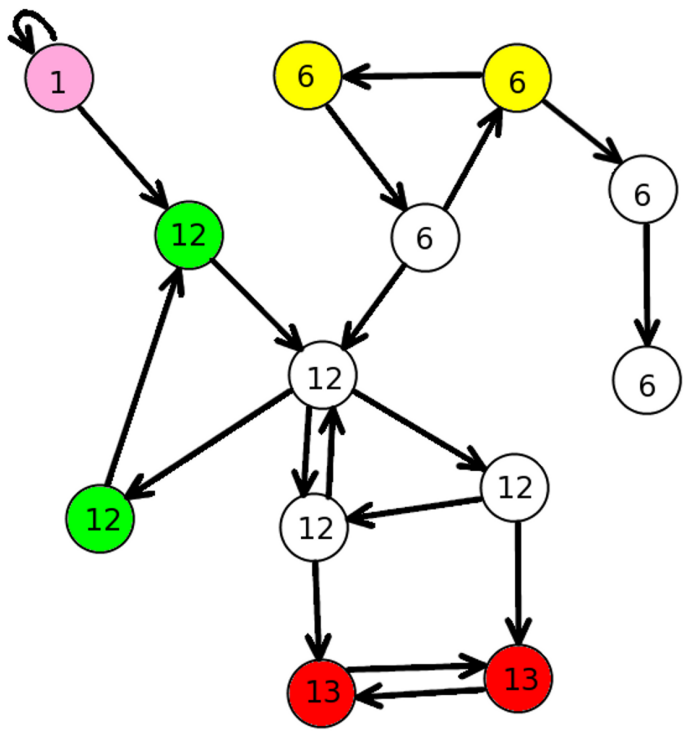
Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC



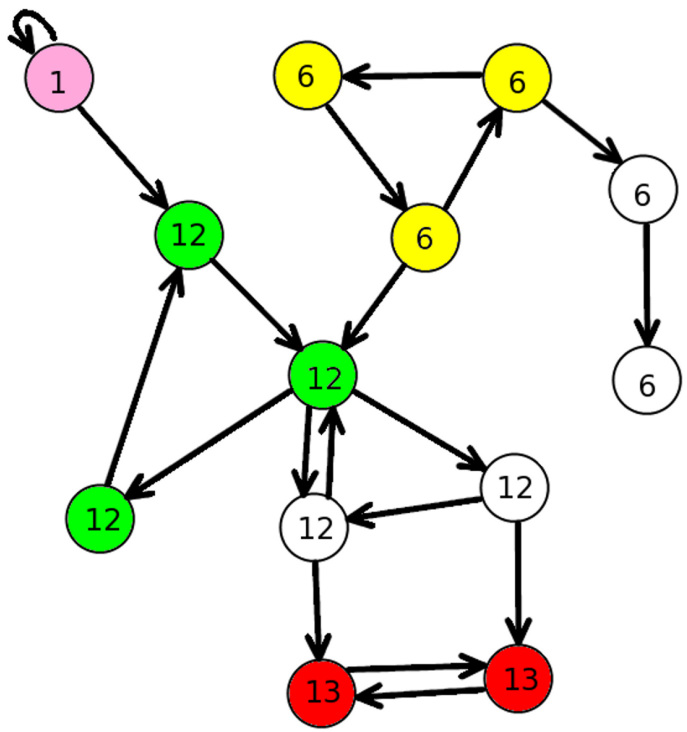
Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices reachable **backward** from each root with the same color



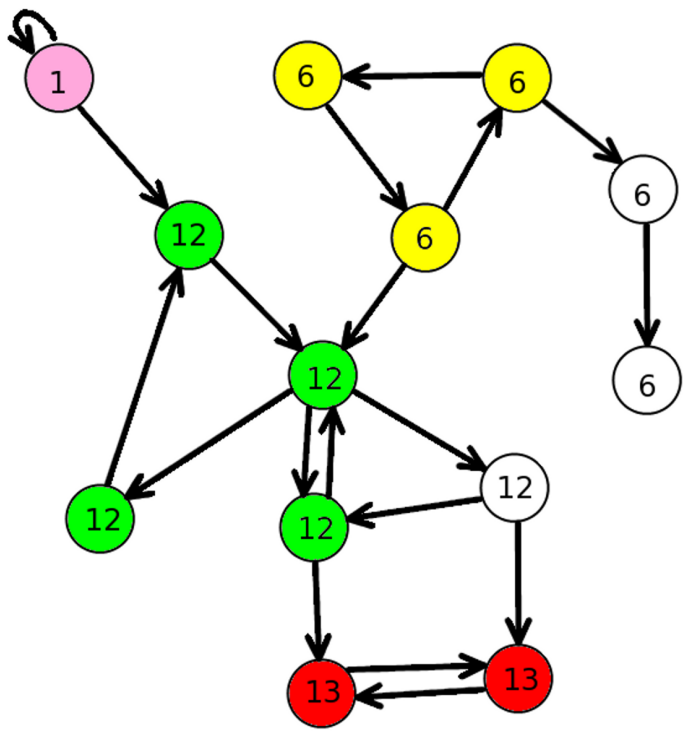
Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices reachable **backward** from each root with the same color



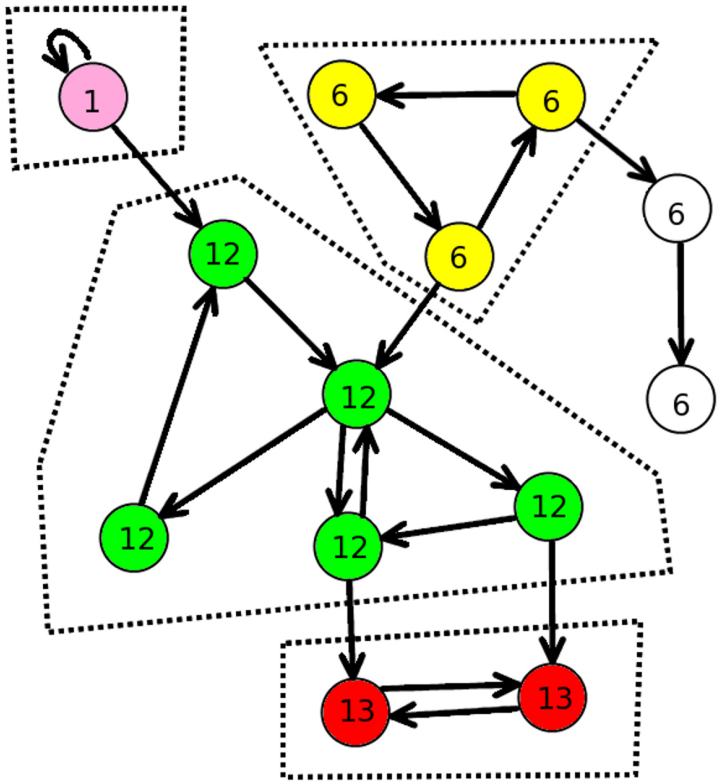
Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices reachable **backward** from each root with the same color



Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices reachable **backward** from each root with the same color

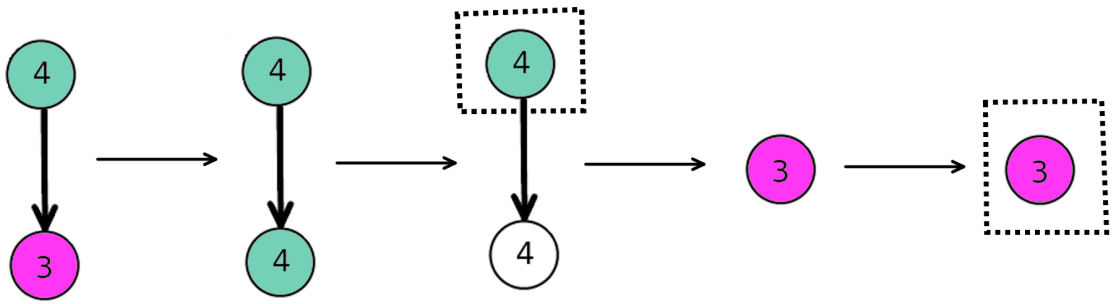


Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices reachable **backward** from each root with the same color
- Remove found SCCs, reset colors, and repeat until no vertices remain

Previous Algorithms
Coloring

- Consider vertex identifiers as *colors*
- Highest colors are propagated **forward** through the network to create sets
- Consider the original vertex of each color to be the *root* of a new SCC
- Each SCC is all vertices reachable **backward** from each root with the same color
- Remove found SCCs, reset colors, and repeat until no vertices remain



Previous Algorithms
Other Previous Work

- Barnat et al. (2011)
  - Evaluated coloring, FW-BW, and several other algorithms running in parallel on CPU and Nvidia CUDA platform
- Hong et al. (2013)
  - Parallel FW-BW with 1 and 2 sized SCC trimming, set partitioning after finding largest SCC based on WCCs, in-house task queue for load balancing

Current Implementation
Observations

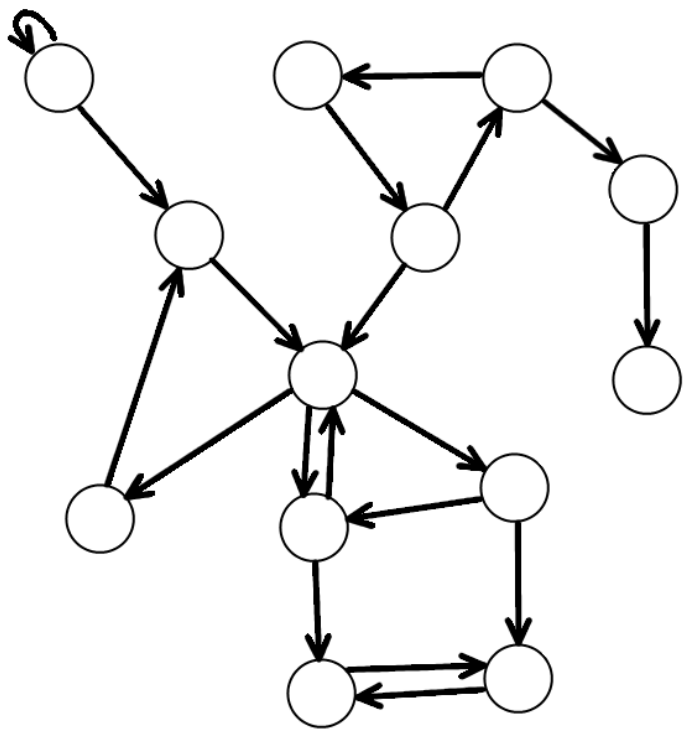
- Most real-world graphs have one giant SCC and many many small SCCs
- FW-BW can be efficient at finding large SCCs, but when there are many small disconnected ones, the remainder set will dominate, creating a large work imbalance
- Coloring is very inefficient at finding a large SCC, but is efficient at finding many small ones
- Tarjan's [6] serial algorithm runs extremely quick for a small number of vertices, scales poorly for a larger number of vertices
- Obvious solution: combine these methods

Current Implementation
Multistep Method

- Do (no/simple/complete) trimming
- Perform single iteration of FW-BW to remove giant SCC
- Do coloring until some threshold of remaining vertices is reached
- Finish with serial algorithm

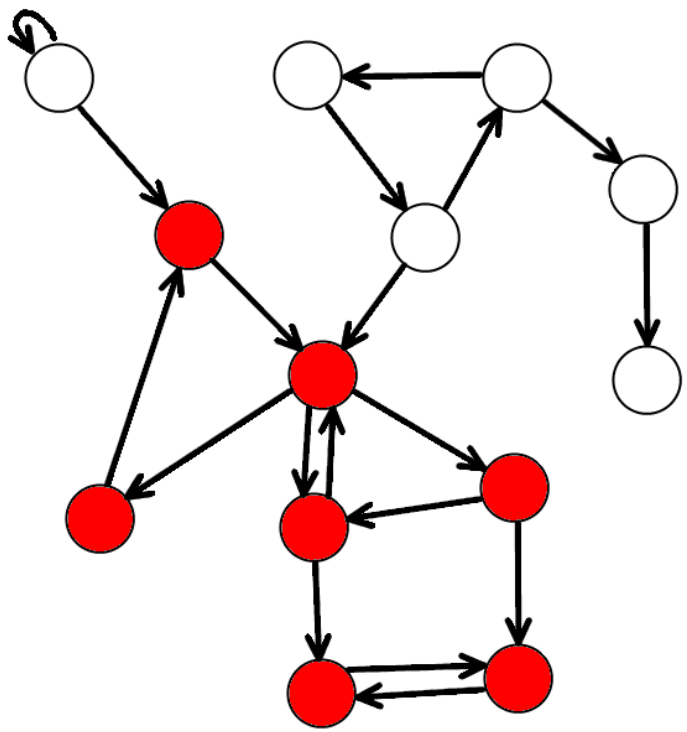
Current Implementation
Multistep Method: FWBW-SCC

- Since we don't care about $(D \setminus S), (P \setminus S), R$ sets, we only need to look for $(S = P \cap D)$



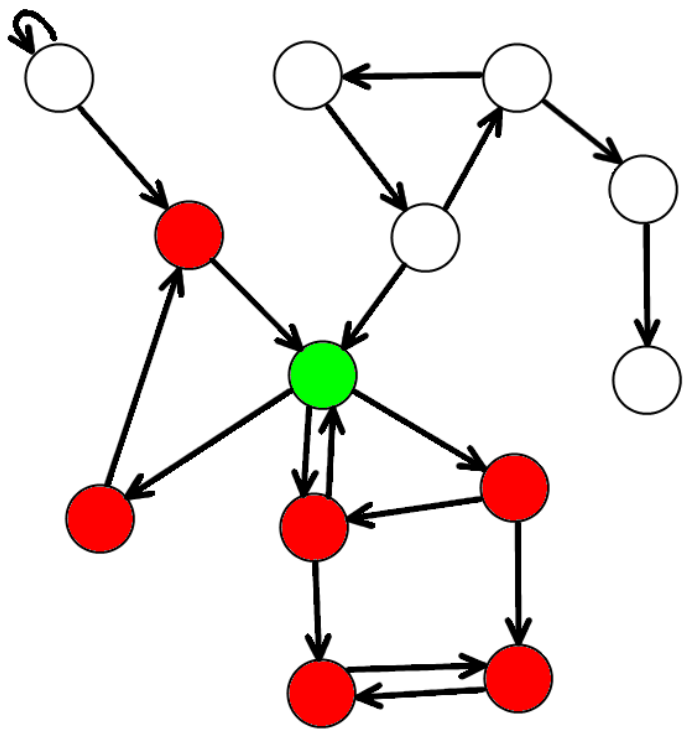
Current Implementation
Multistep Method: FWBW-SCC

- Since we don't care about $(D \setminus S), (P \setminus S), R$ sets, we only need to look for $(S = P \cap D)$
- Begin as before, select pivot and find all of (D)



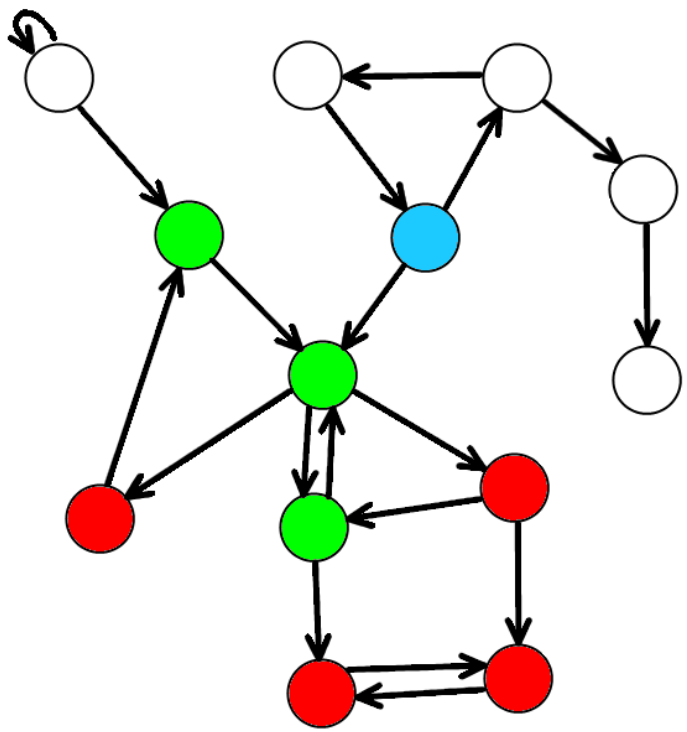
Current Implementation
Multistep Method: FWBW-SCC

- Since we don't care about $(D \setminus S), (P \setminus S), R$ sets, we only need to look for $(S = P \cap D)$
- Begin as before, select pivot and find all of (D)
- For backward search, only consider vertices already marked in (D)



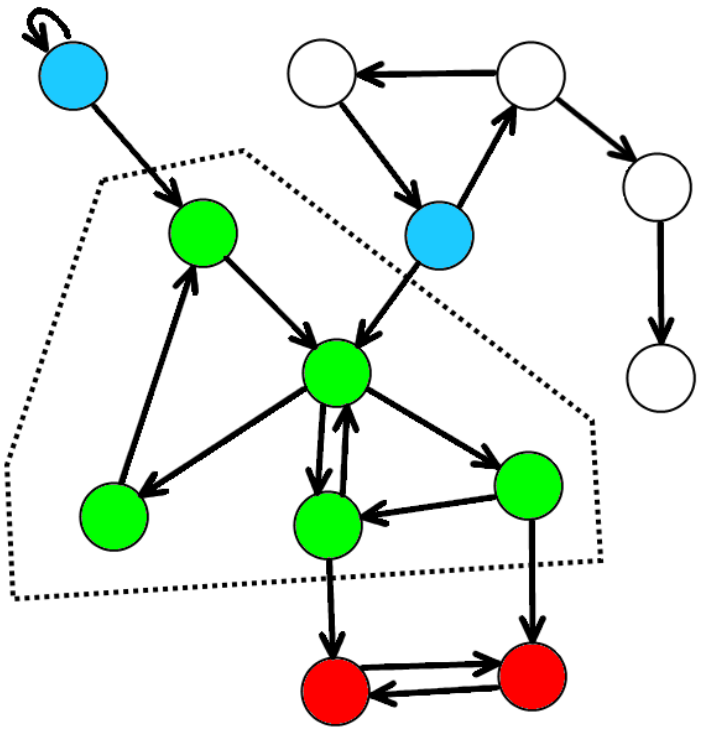
Current Implementation
Multistep Method: FWBW-SCC

- Since we don't care about $(D \setminus S), (P \setminus S), R$ sets, we only need to look for $(S = P \cap D)$
- Begin as before, select pivot and find all of (D)
- For backward search, only consider vertices already marked in (D)



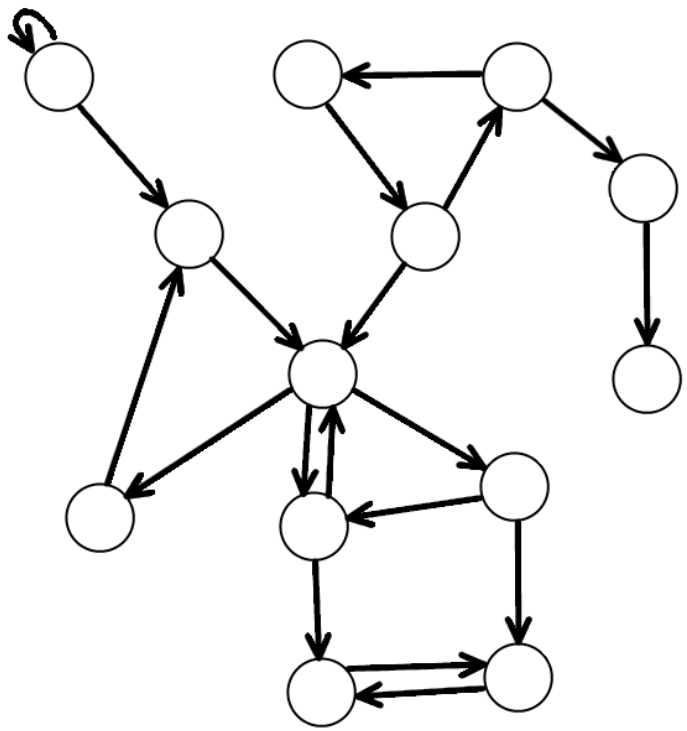
Current Implementation
Multistep Method: FWBW-SCC

- Since we don't care about $(D \setminus S), (P \setminus S), R$ sets, we only need to look for $(S = P \cap D)$
- Begin as before, select pivot and find all of (D)
- For backward search, only consider vertices already marked in (D)
- For certain graphs, this can dramatically decrease the search space



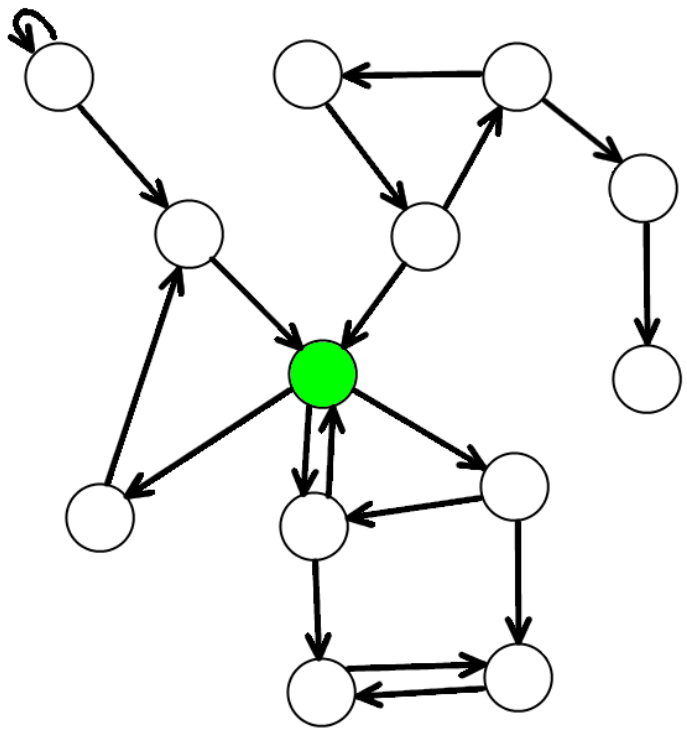
Current Implementation
Multistep Method: Hybrid Search

- We can also combine the forward and backward search



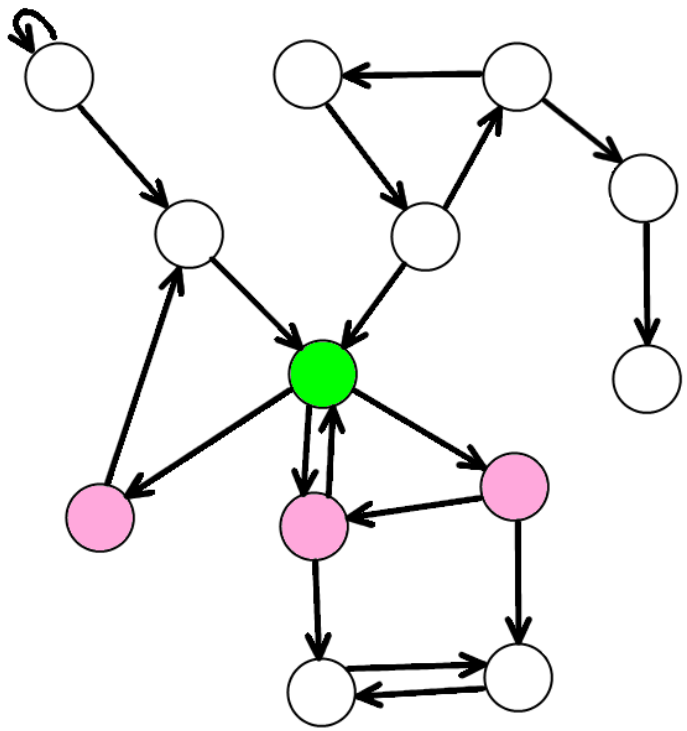
Current Implementation
Multistep Method: Hybrid Search

- We can also combine the forward and backward search
- As we search forward, if we find any vertex already marked as in the SCC, we know the current vertex must be as well



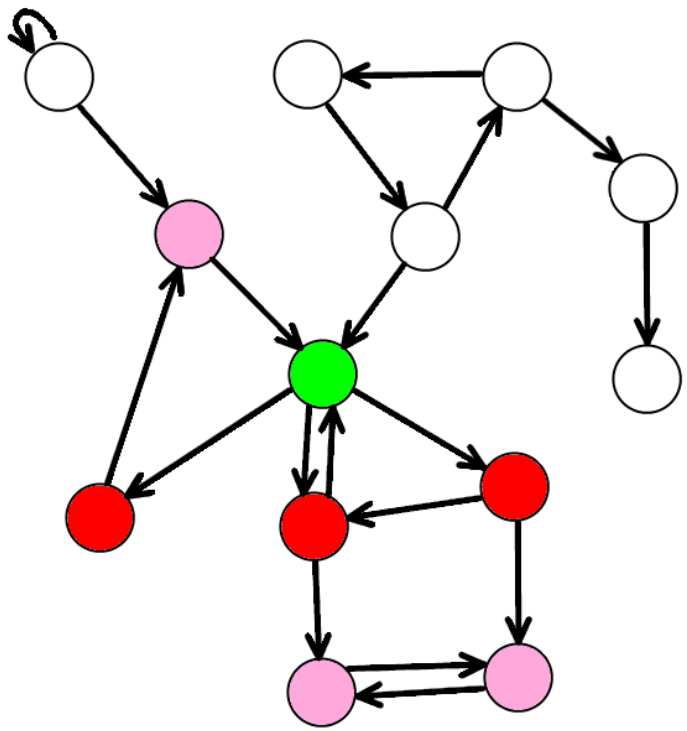
Current Implementation
Multistep Method: Hybrid Search

- We can also combine the forward and backward search
- As we search forward, if we find any vertex already marked as in the SCC, we know the current vertex must be as well



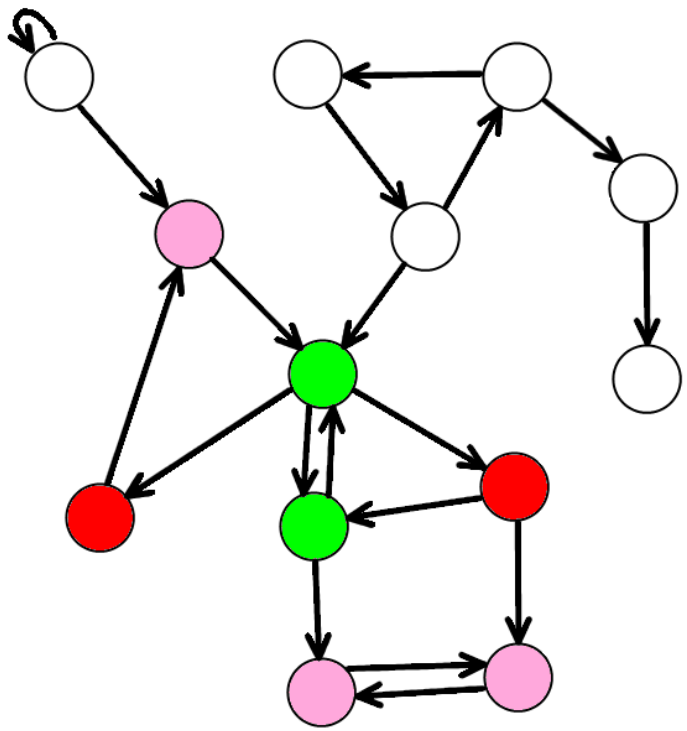
Current Implementation
Multistep Method: Hybrid Search

- We can also combine the forward and backward search
- As we search forward, if we find any vertex already marked as in the SCC, we know the current vertex must be as well



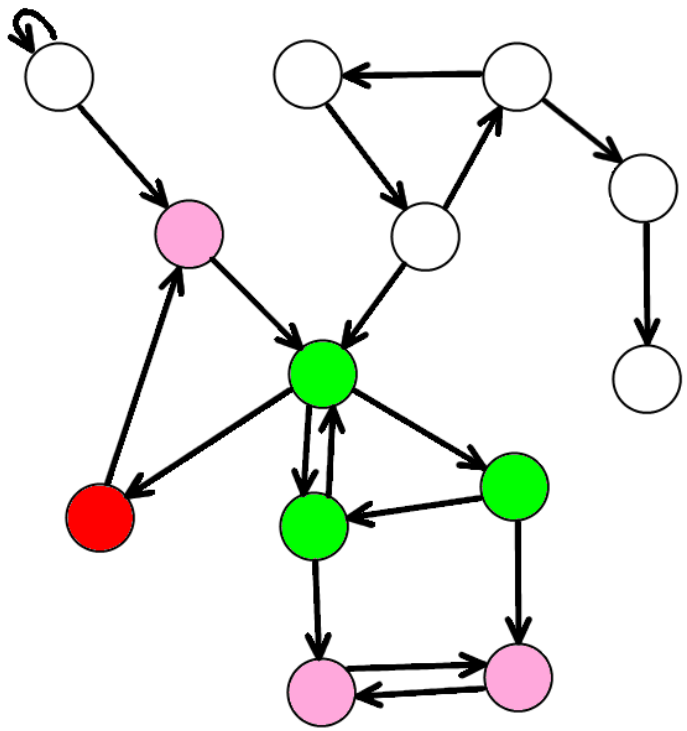
Current Implementation
Multistep Method: Hybrid Search

- We can also combine the forward and backward search
- As we search forward, if we find any vertex already marked as in the SCC, we know the current vertex must be as well
- We mark the current vertex and all of its already visited predecessors as in the SCC



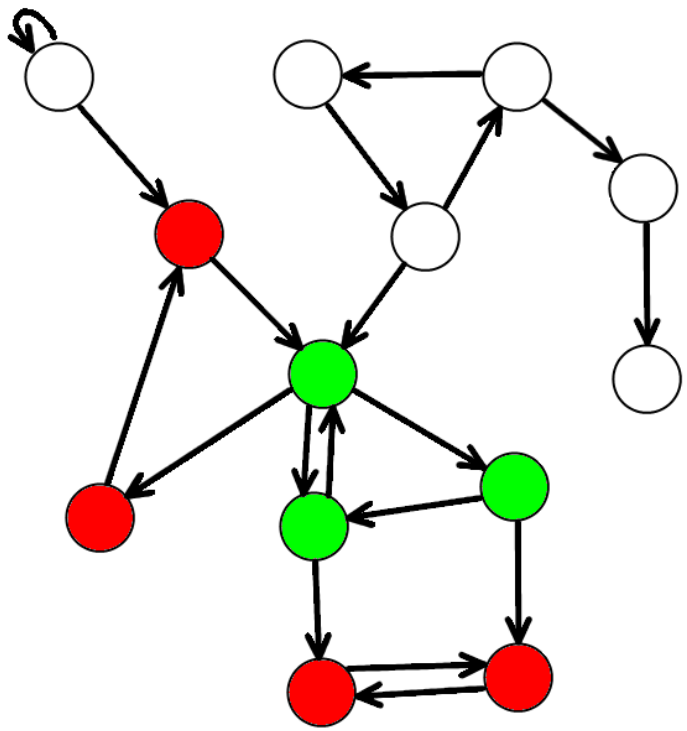
Current Implementation
Multistep Method: Hybrid Search

- We can also combine the forward and backward search
- As we search forward, if we find any vertex already marked as in the SCC, we know the current vertex must be as well
- We mark the current vertex and all of its already visited predecessors as in the SCC



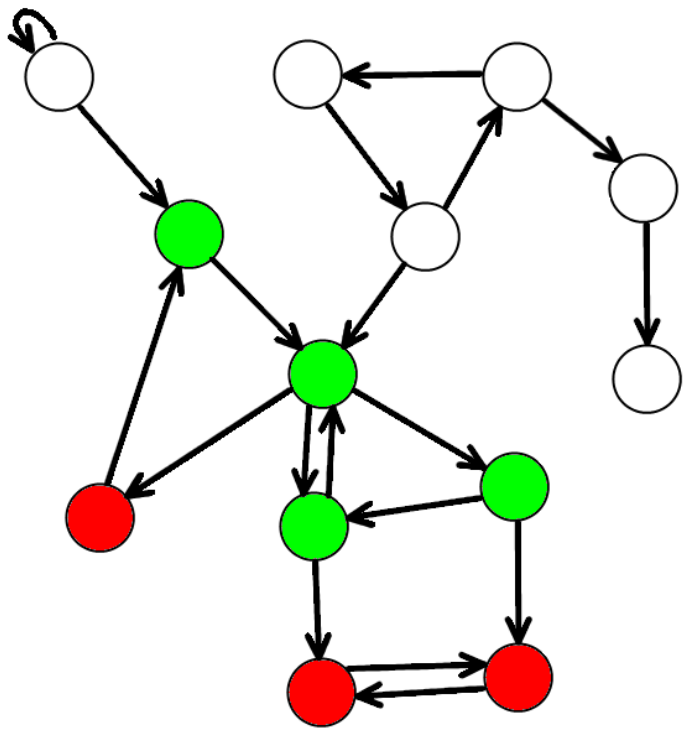
Current Implementation
Multistep Method: Hybrid Search

- We can also combine the forward and backward search
- As we search forward, if we find any vertex already marked as in the SCC, we know the current vertex must be as well
- We mark the current vertex and all of its already visited predecessors as in the SCC



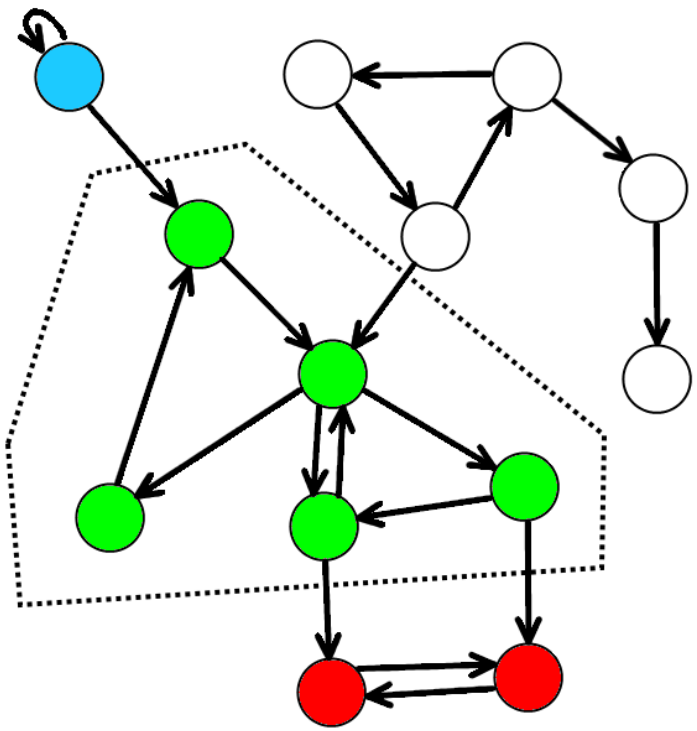
Current Implementation
Multistep Method: Hybrid Search

- We can also combine the forward and backward search
- As we search forward, if we find any vertex already marked as in the SCC, we know the current vertex must be as well
- We mark the current vertex and all of its already visited predecessors as in the SCC



Current Implementation
Multistep Method: Hybrid Search

- We can also combine the forward and backward search
- As we search forward, if we find any vertex already marked as in the SCC, we know the current vertex must be as well
- We mark the current vertex and all of its already visited predecessors as in the SCC



Performance Results
Test Algorithms

- **Multistep**: Simple trimming, parallel DFS FWBW-SCC, coloring until less than 100k vertices remain, serial Tarjan
- **Hybrid**: Simple trimming, parallel DFS hybrid search, coloring until less than 100k vertices remain, serial Tarjan
- **FW-BW**: Complete trimming, FW-BW algorithm until completion
- **Coloring**: Complete trimming, Coloring until 100K vertices, serial Tarjan
- **Serial**: Serial Tarjan

Performance Results

Test Environment and Networks

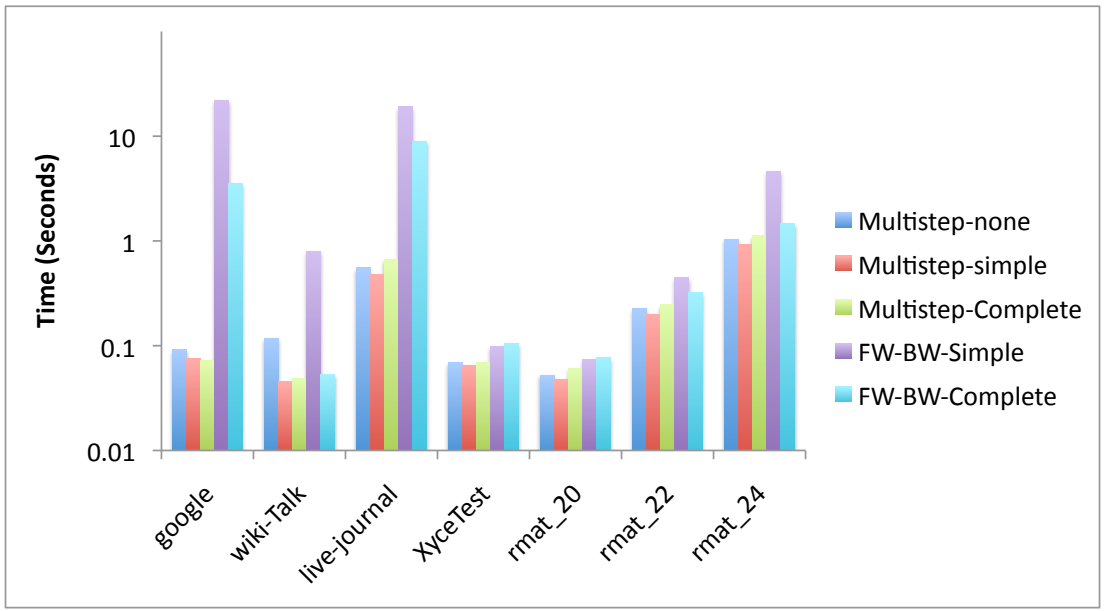
- Vesper (AMD): 64 Magnycour cores, 8x8 configuration.
- Compton (Intel): Xeon E5-2670 (Sandybridge), dual socket, 16 cores.

Network	n	m	d_{avg}	d_{max}	Dia.	# SCCs	max SCC
Friendster	66M	1.8B	53	5.2K	25	3M	63M
LiveJournal	4.8M	69M	14	20K	18	970K	3.8M
Wiki-Talk	2.4M	5.0M	2.1	100K	9	2.2M	500K
USA Road Net	24M	29M	1.2	9	~8K	1	24M
Patents	3.8M	17M	4.4	770	22	3.8M	1
Google Web	876K	5.1M	5.8	460	22	410K	430K
XyceTest	1.9M	8.3M	4.2	246	~91	400K	1.5M
R-MAT 20	560K	8.4M	15	24K	9	210K	360K
R-MAT 22	2.1M	34M	16	60K	9	790K	1.3M
R-MAT 24	7.7M	130M	17	150K	9	3.0M	4.7M

Graphs used in experiments retrieved from [7, 8, 9].

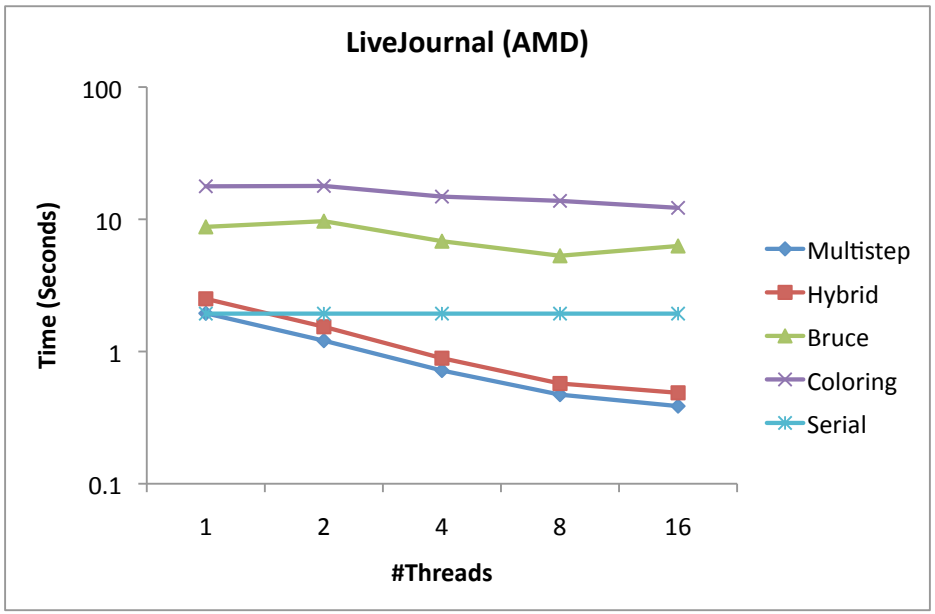
Performance Results
Trimming Options

- Doing complete trimming isn't always the best choice for multistep; sometimes even no trimming is fastest; extra trimming work is handled better by coloring or serial algorithm
- Complete is almost always the best choice when doing FW-BW (no trimming not run with FW-BW due to excessive processing times)



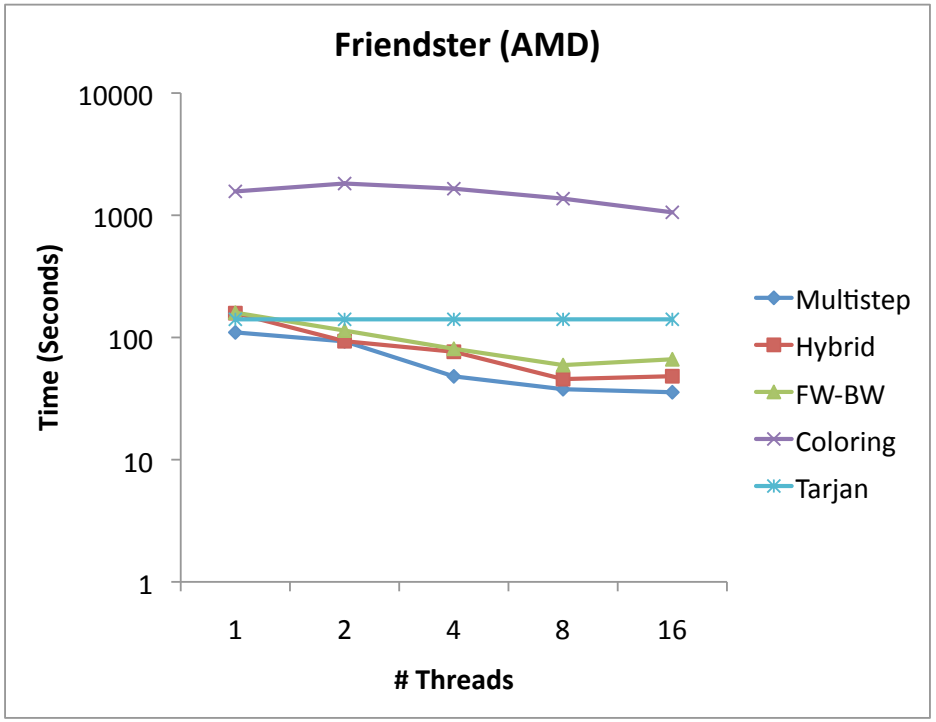
Performance Results
LiveJournal on AMD

- Multistep and hybrid show good (5×) parallel scaling
- FW-BW algorithm suffers from previously mentioned load imbalance
- Relatively high diameter (18) results in poor coloring performance



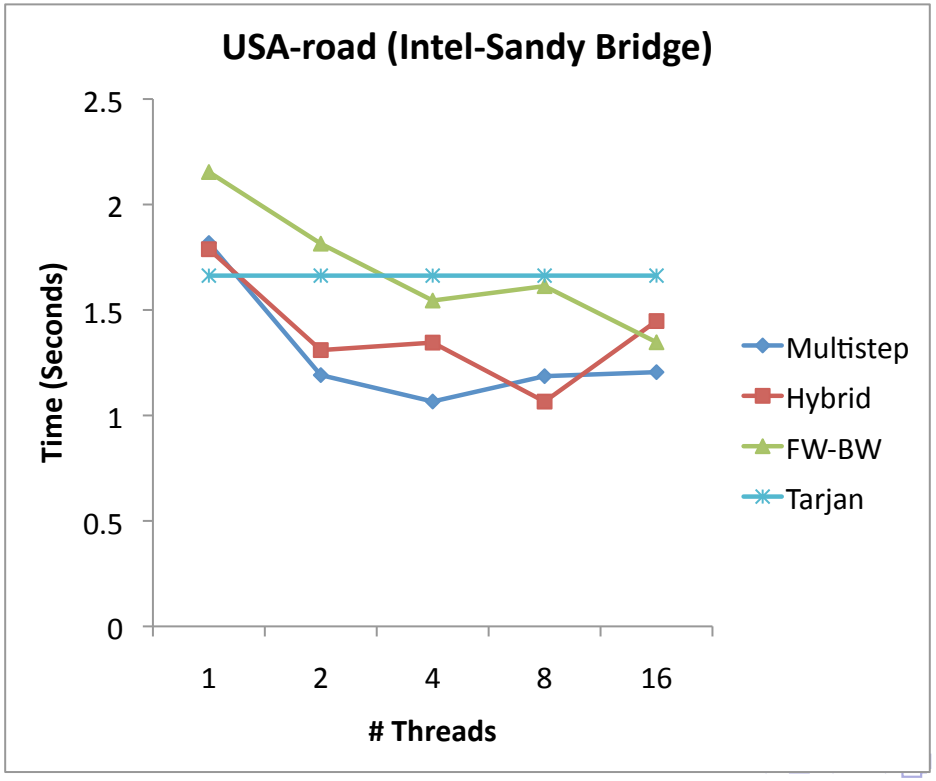
Performance Results
Friendster on AMD

- About (4×) parallel scaling for multistep
- FW-BW and multistep/hybrid algorithms similar due to small number of SCCs remaining after initial trim (∼70)
- Very poor coloring performance from high diameter and large size



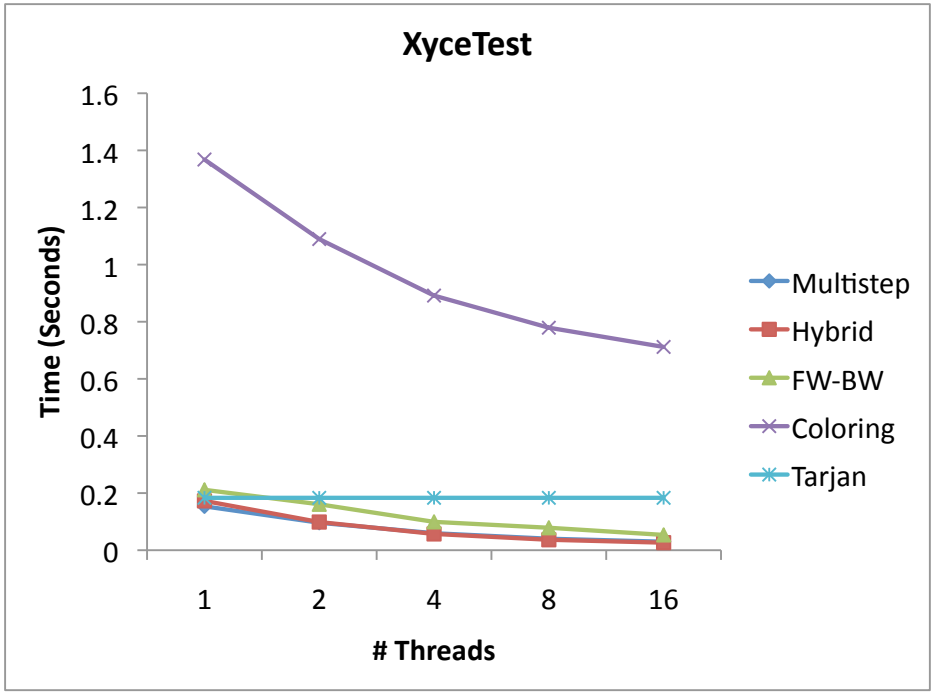
Performance Results
USA Road Net on Intel

- Coloring not shown in results due to very poor performance (over 1000s)
- Similar performance between multistep, hybrid, and FW-BW due to graph being fully connected; all three algorithms explore the entire graph
- Although not much speedup, all three are still faster than Tarjan's



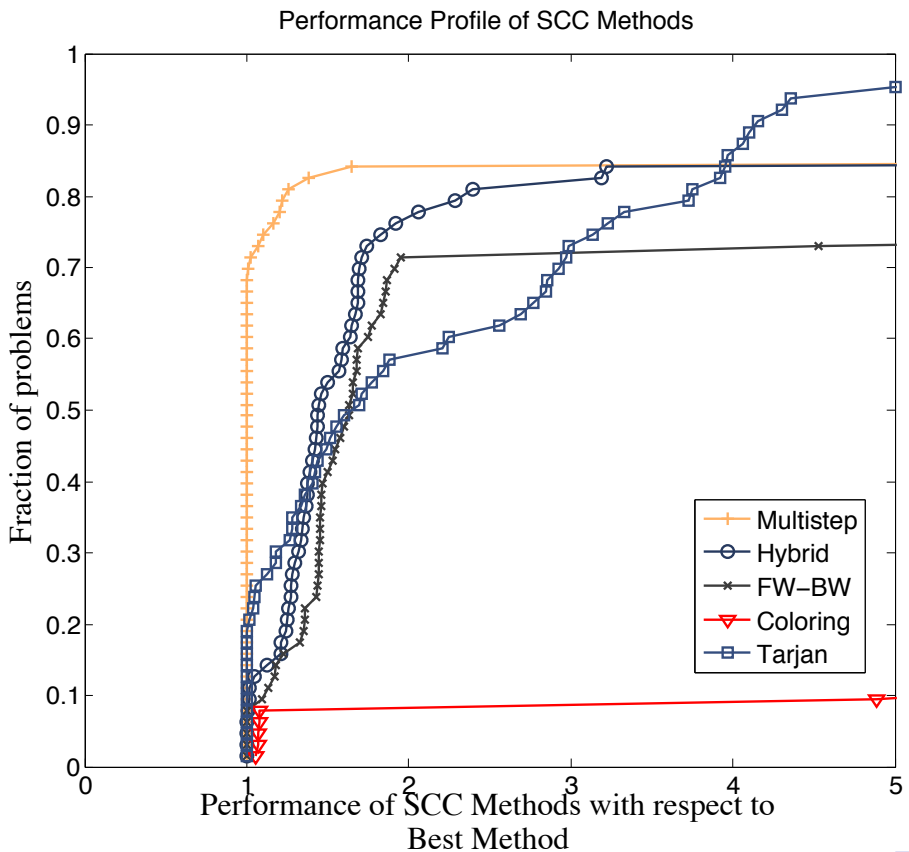
Performance Results
XyceTest on Intel

- Good speedup for all tested algorithms ($\times 7$ max)
- Hybrid algorithm fastest over multistep by $\sim 10\%$
- Both hybrid and multistep are faster than Tarjan's with a single thread



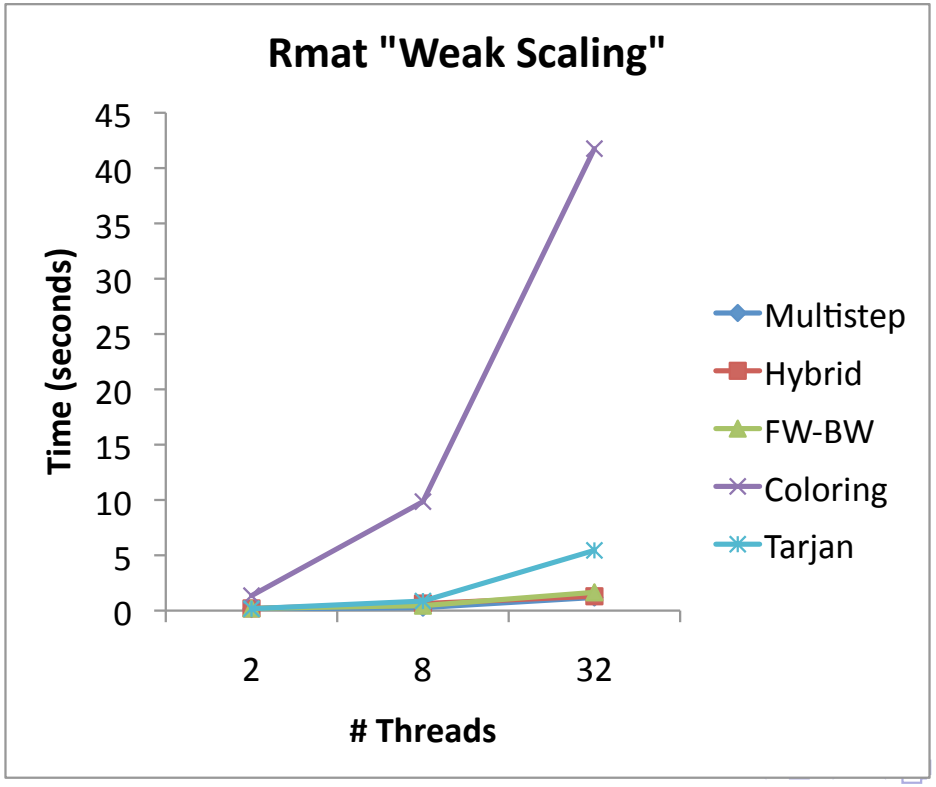
Performance Results
Performance Profile

- 70% of the time multistep is the best approach
- Coloring by itself is the worst algorithm
- FW-BW is within 2× of best algorithm in 70% of the problems



Performance Results
R-MAT Weak Scaling

- R-MAT graphs (20,22,24): n, m, number of SCCs, and size of max SCC all increase by $\sim 4\times$ with each graph
- Approximate weak scaling observed with FW-BW and multistep/hybrid
- Scaling much worse for Tarjan's and much much worse for coloring



Bibliography

[1] W. McLendon III, B. Hendrickson, and S. J. Plimpton, "Finding strongly connected components in distributed graphs," *Lecture Notes in Computer Science*, vol. 1800, pp. 505–512, 2000.

[2] L. K. Fleischer, B. Hendrickson, and A. Pinar, "On identifying strongly connected components in parallel," *Parallel and Distributed Processing*, vol. 65, pp. 901–910, 2005.

[3] S. Orzan, "On distributed verication and veried distribution," Ph.D. dissertation, Free University of Amsterdam, 2004.

[4] S. Hong, N. C. Rodia, and K. Olukotun, "Technical report: On fast parallel detection of strongly connected components (scc) in small-world graphs," Stanford University, Tech. Rep., 2013.

[5] J. Barnat, P. Bauch, L. Brim, and M. Cevska, "Computing strongly connected components in parallel on cuda," in *Parallel and Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 544–555.

[6] R. E. Tarjan, "Depth first search and linear graph algorithms," *SIAM Journal of Computing*, vol. 1, pp. 146–160, 1972.

[7] J. Leskovec, "SNAP: Stanford network analysis project," http://snap.stanford.edu/index.html, last accessed 3 July 2013.

[8] DIMACS, "9th dimacs implementation challenge - shortest paths."

[9] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *In SDM*, 2004.

Conclusions and Future work

- Current individual algorithms have drawbacks, using multistep/hybrid approach exploits strengths and minimizes weaknesses
- Parallel multistep shows good speedup compared to serial Tarjan, sometimes is faster than Tarjan with a single thread
- For most large real world graphs, a majority of the time is spent in initial SCC search, best way to decrease computation times is with a better hardware optimized BFS/DFS search
- Further optimizations to the FWBW-SCC and FWBW-hybrid search algorithms may be possible