Provably Fast and Space-Efficient Parallel Biconnectivity (PPoPP'23 Best Paper)

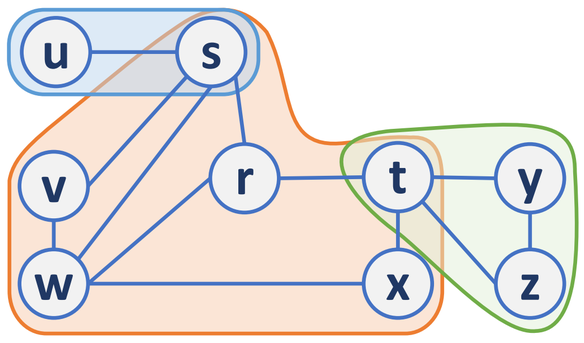
Xiaojun Dong Letong Wang Yan Gu Yihan Sun

RIVERSIDE

Full Version: https://arxiv.org/abs/2301.01356 Code: https://github.com/ucrparlay/FAST-BCC

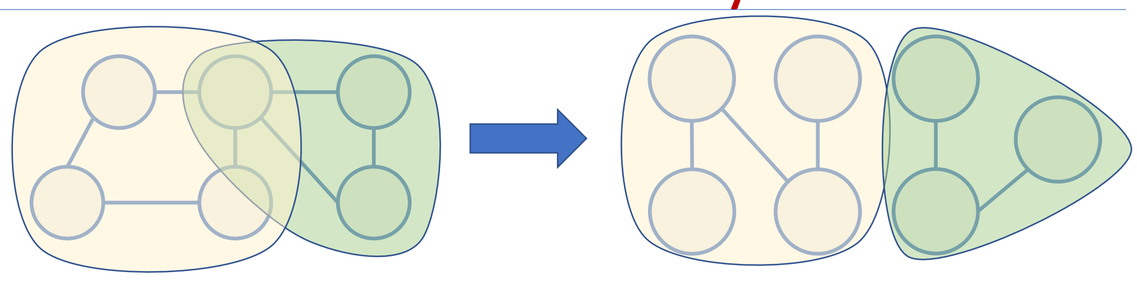
Problem Definitions

- Given an undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges
- A ***connected component (CC)*** is a maximal subset in V such that every two vertices in it are connected by a path



- Two vertices are biconnected if they are connected and remain connected after removing any other single vertex
- A ***biconnected component (BCC)*** (or a block) is a maximal subset of biconnected vertices

The Skeleton-Connectivity Framework



$G = (V, E)$ ⟺ "**Skeleton**" $G' = (V', E')$

Biconnectivity of G ⟺ Connectivity of G'

(some postprocessing may be needed)

- Many BCC algorithms follow the skeleton-connectivity framework
- Challenge: generating the skeleton (time-)efficiently and storing the skeleton (space-)efficiently!

Related Work

Algorithms	Generating Skeleton	Storing Skeleton
Sequential Hopcroft-Tarjan [HT73]	**DFS Tree** × Hard to parallelize!	Not stored (processed on-the-fly) ✓ Space-efficient
Parallel Tarjan-Vishkin [TV85]	**Arbitrary Spanning Tree** ✓ Work- and span-efficient!	Skeleton needs $O(m)$ extra space × Space-inefficient!
Existing Parallel Implementations [CB05, DBS18, SM14]	**BFS Tree** (Span proportional to graph diameters!) × Low parallelism in the worst case	$O(n)$ space, or maintained implicitly in $O(n)$ space ✓ Space-efficient
Our algorithm (FAST-BCC)	**Arbitrary Spanning Tree** ✓ Work- and span-efficient!	Maintained implicitly using $O(n)$ space ✓ Space-efficient

Experimental Results

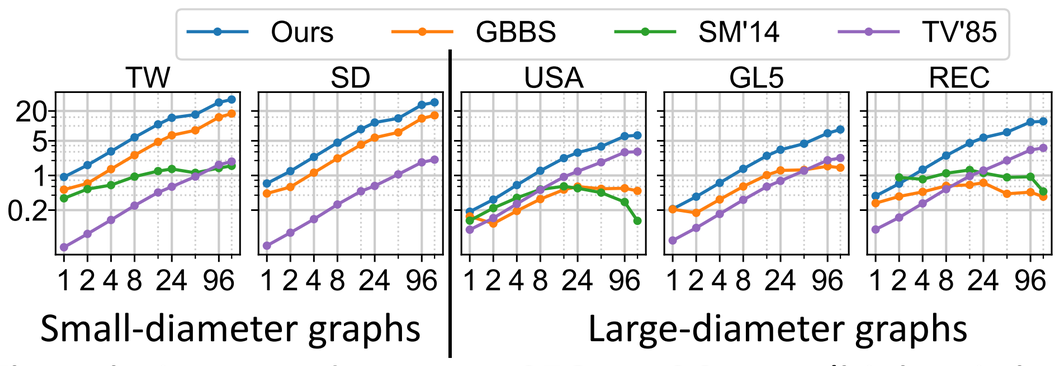
		Ours	GBBS	SM'14	SEQ
Social	YT	5.88	4.36	3.15	1.00
	OK	30.51	19.91	5.66	1.00
	LJ	17.92	11.77	n	1.00
	TW	34.21	17.42	2.40	1.00
	FT	39.26	18.93	10.22	1.00
	MEAN	21.23	12.75	4.57	1.00
Web	GG	8.92	5.65	n	1.00
	SD	29.74	16.46	n	1.00
	CW	30.37	17.52	n	1.00
	HL14	32.46	19.96	n	1.00
	HL12	33.99	29.15	n	1.00
	MEAN	24.53	15.68	-	1.00
Road	CA	5.15	0.55	n	1.00
	USA	6.69	0.49	0.60	1.00
	GE	10.77	1.43	2.44	1.00
	MEAN	7.18	0.73	1.21	1.00
K-NN	HH5	7.01	1.14	n	1.00
	CH5	4.11	0.37	n	1.00
	GL2	6.24	1.64	n	1.00
	GL5	8.53	1.44	n	1.00
	GL10	10.59	4.31	n	1.00
	GL15	11.88	5.91	n	1.00
	GL20	11.84	6.88	n	1.00
	COS5	14.16	6.86	n	1.00
	MEAN	8.68	2.42	-	1.00
Synthetic	SQR	18.50	1.59	10.56	1.00
	REC	12.48	0.36	3.02	1.00
	SQR'	8.06	0.85	n	1.00
	REC'	7.81	0.48	n	1.00
	Chn7	11.97	0.04	0.08	1.00
	Chn8	11.97	0.04	0.06	1.00
	MEAN	11.30	0.27	0.18	1.00
TOTAL MEAN		12.89	2.50	0.96	1.00

0 .5 1 2 4 8 16 32 >32

n = no support

MEAN = geometric mean

- The heatmap shows relative speedups for parallel BCC algorithms over the sequential Hopcroft-Tarjan algorithm (SEQ) using 96 cores (192 hyper-threads)
- On large-diameter graphs, GBBS and SM'14 only achieve 0.18-2.42x speedup. Can be slower than SEQ!
- FAST-BCC achieves the best performance on all graphs
- On average, it is significantly faster than all baselines!
  - About 13x faster than SEQ
  - About 5x faster than GBBS (previous best implementation)



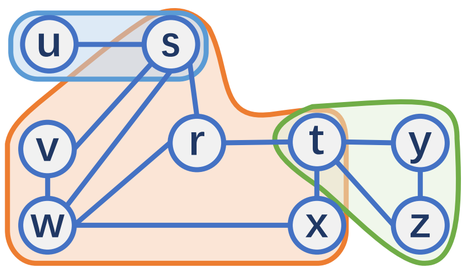
- The relative speedups over SEQ on 96-core (higher is better)
- GBBS and SM'14 don't scale well on large-diameter graphs
- FAST-BCC and Tarjan-Vishkin achieve almost-linear scalability on all graphs

References

[1] Guojing Cong and David Bader. 2005. An experimental study of parallel biconnected components algorithms on symmetric multiprocessors (SMPs). In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE.
[2] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. 2021. Theoretically efficient parallel graph algorithms can be fast and scalable. *ACM Transactions on Parallel Computing (TOPC)* 8, 1 (2021), 1–70.
[3] Xiaojun Dong, Letong Wang, Yan Gu, and Yihan Sun. 2023. Provably Fast and Space-Efficient Parallel Biconnectivity. *ACM Symposium on Principles and Practice of Parallel Programming (PPOPP)* (2023), 52–65.
[4] John Hopcroft and Robert Tarjan. 1973. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM* 16, 6 (1973), 372–378.
[5] George Slota and Kamesh Madduri. 2014. Simple parallel biconnectivity algorithms for multicore platforms. In *IEEE International Conference on High Performance Computing (HiPC)*. IEEE, 1–10.
[6] Robert E Tarjan and Uzi Vishkin. 1985. An efficient parallel biconnectivity algorithm. *SIAM J. on Computing* 14, 4 (1985), 862–874.

Algorithm Overview

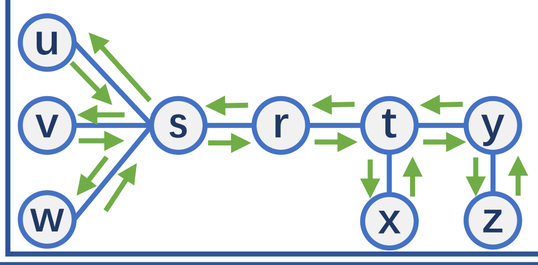
Input Graph G: contains 3 BCCs {s, u}, {r, s, t, v, w, x}, {t, y, z}.



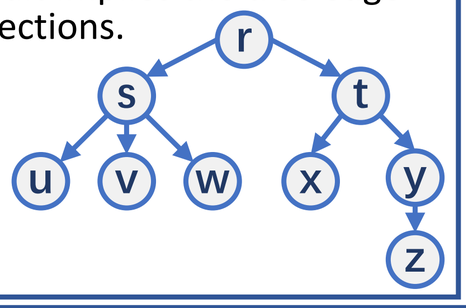
Step 1: First CC. Find the CCs of G and a spanning tree (forest).

Step 2: Rooting. Generate rooted spanning trees.

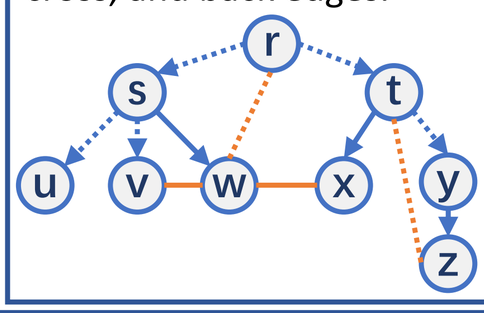
Step 2.1: Based on the spanning tree T of G. Create the linked list of the Euler tour of T.



Step 2.2: Set any vertex as the root and run list ranking. The result implies the tree edge directions.



Step 3: Tagging.
Compute tags (first/last/low/high/…) for each vertex. Use these tags to identify fence, plain, cross, and back edges.

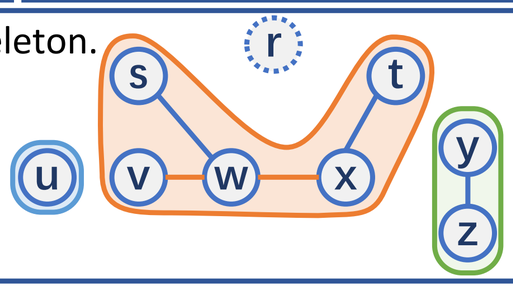


Tree edge:
Fence edge
Plain edge

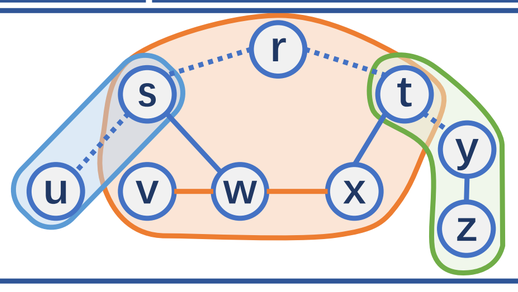
Non-tree edge:
Back edge
Cross edge

Step 4: Last CC. Run CC on the skeleton.

Step 4.1: Find the CCs of the skeleton G' only with cross and plain edges in G (solid edges in Step 3). Ignore the root.



Step 4.2: Assign the component head to each CC in G'. Each CC in G' with its component head is a BCC.



BCC1 {s, u}:
Head s + {u}
BCC2 {s, t, v, w, x}:
Head r + {s, t, v, w, x}
BCC3 {t, y, z}:
Head t + {y, z}