A Comparison of Parallel Algorithms for Connected Components

John Greiner
School of Computer Science
Carnegie Mellon University
jdg@cs.cmu.edu

Previous Theoretical Work

Previous CRCW work includes

Algorithm	Work (general):	Work (grids):	Time:	Randomized
Shiloach-Vishkin ’82	$m \log n$		$\log n$	No
Awerbuch-Shiloach ’83				No
Reif ’85				Yes
Cole-Vishkin ’86	$m\ \alpha(m,n)(m+n)$			No
Gazit ’91 (optimal)	$m+n$			Yes
Random Mate	$m \log n$	m	$\log^2 n$	Yes
Hybrid	$(m+n) \log n$	$m + n \log n$		No

m edges, n nodes

CREW, EREW: Hirschberg-Chandra-Sarwate ’79, Johnson-Metaxas ’91, Karger-Nisan-Parnas ’92, Chong-Lam ’93, Halperin-Zwick ’94, ...

Previous Pragmatic Work

Little general-purpose empirical work

- Physics and vision — most algorithms specialized for grids
- Pardalos-Rentala ’89 — EREW algorithm on IBM 3090-400E
- Woo-Sahni ’89 — hypercube algorithm on NCUBE

Empirical comparison of multiple general-purpose algorithms

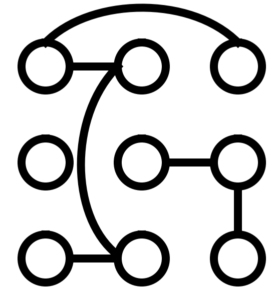
- Coded in NESL — Provides hardware-independent PRAM + scan model
- Tested with multiple classes of graphs
- Executed on multiple architectures

Hybrid as fast as previous algorithms specialized for grids

- On 150 x 150 x 150 grid, 40% of edges:
- Fink-Baden-Jansen ’94 on 256 proc. Paragon: 0.9 sec.
- Hybrid on Cray: 1.6 sec. — New: 0.54 sec. (1 proc.), 0.15 sec. (4 proc.)

Classes of Graphs Used in Testing

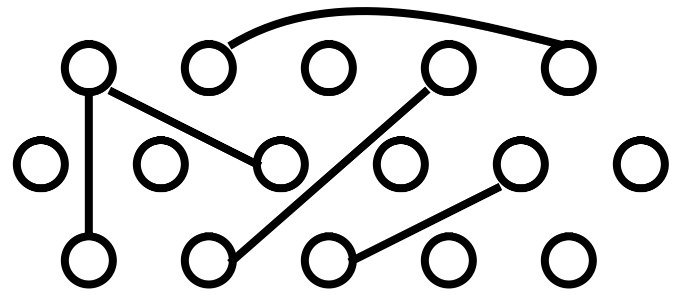
Grids, 2D & 3D



Commonly used in practice

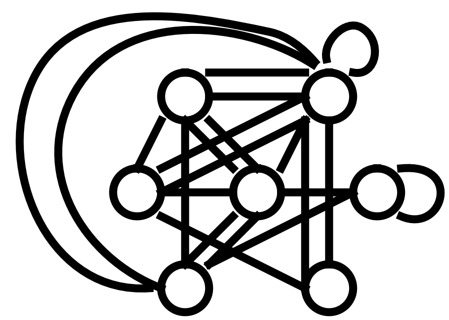
30% or 60% of 2D edges, 20% or 40% of 3D edges

Random — 2% of edges from complete graph



Large graphs increasingly dense

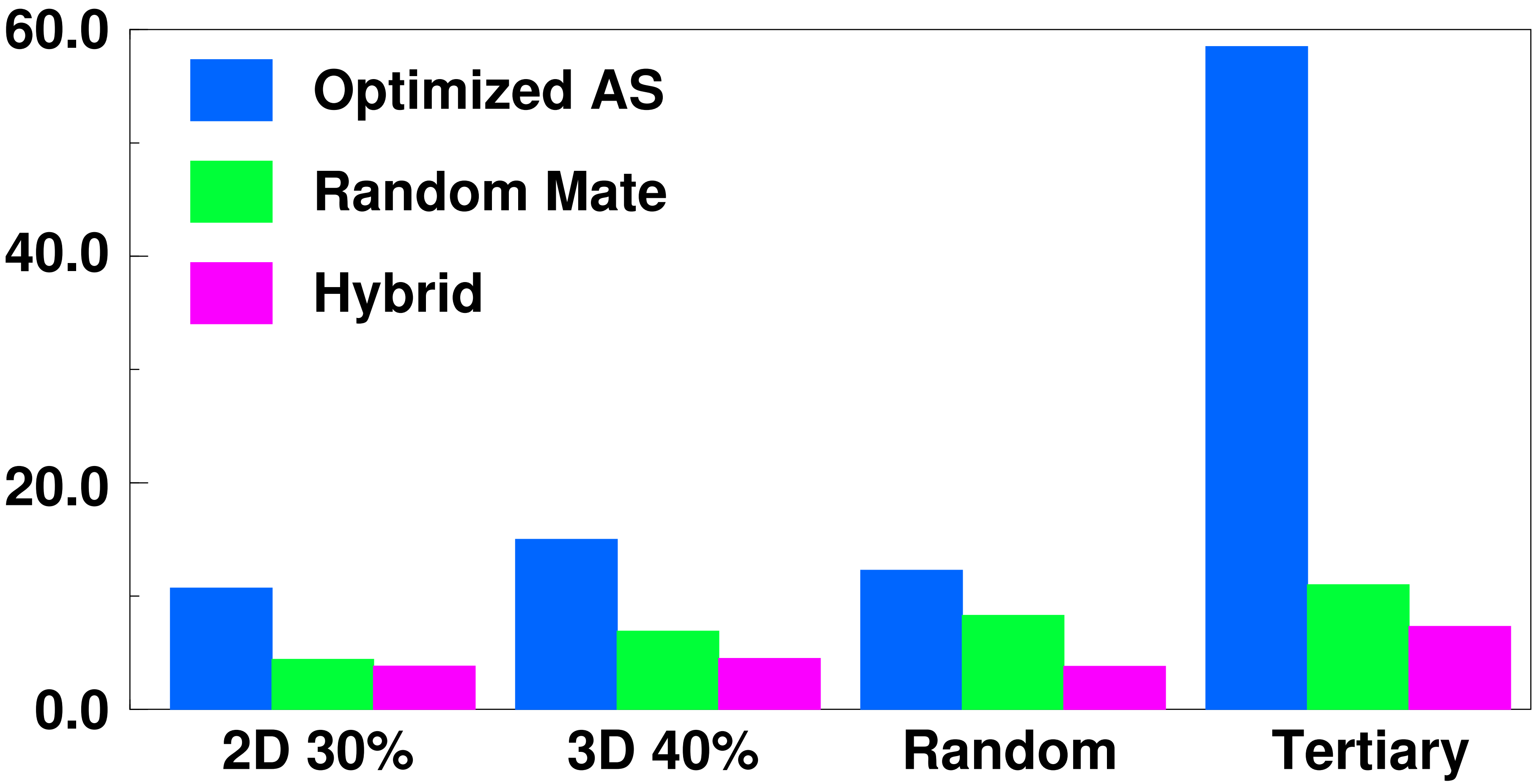
Tertiary — each node picks 3 targets for edges



Somewhat intermediate case

Size of separators affects rate of contraction

Comparison of Algorithms



Times (sec.) on 8K proc. CM-2, graphs of 500,000 edges

Some Ideas in CC Algorithms

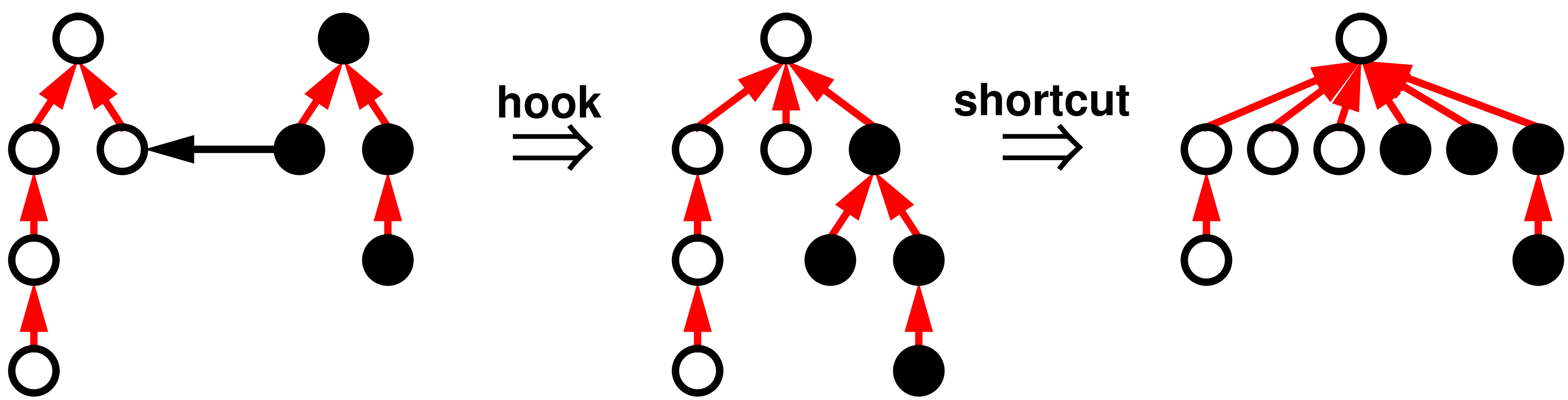
Trees of nodes — known to be in same CC — roots as representatives of trees

Shortcutting — halves height of trees

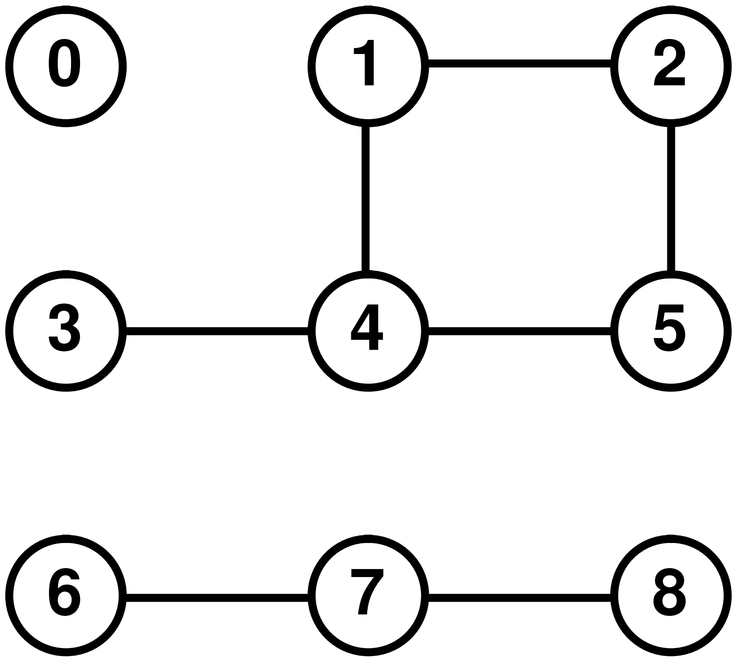
Hooking — combines pairs of trees — Must avoid creating cycles.

Hooking with subsets of edges is key for Gazit's optimality

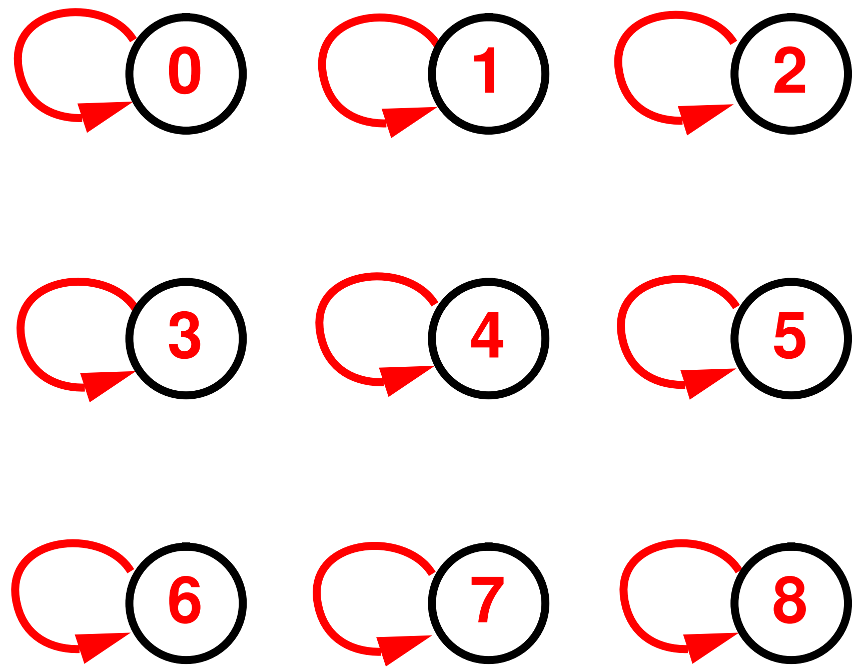
Contracting — removes nodes and/or edges



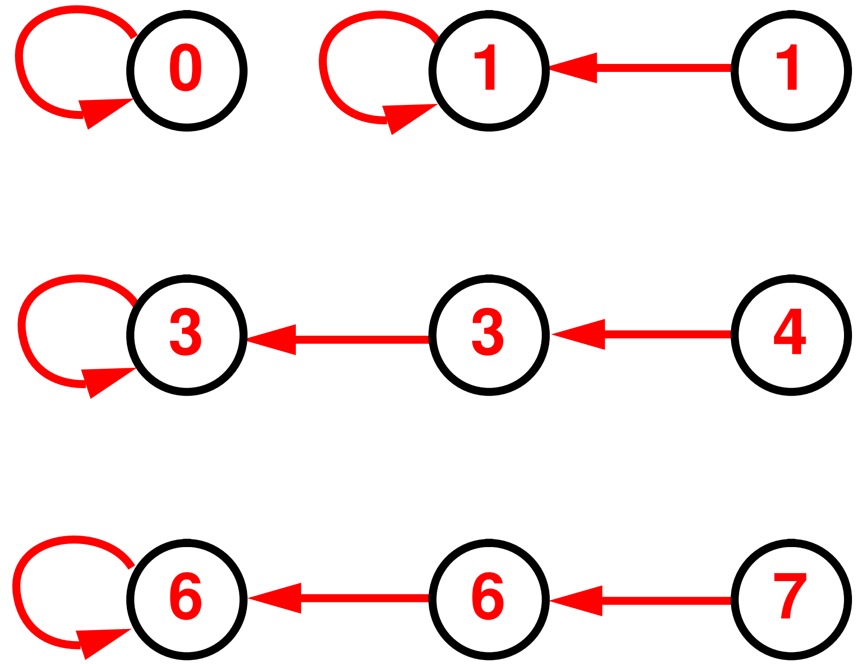
Hybrid Algorithm



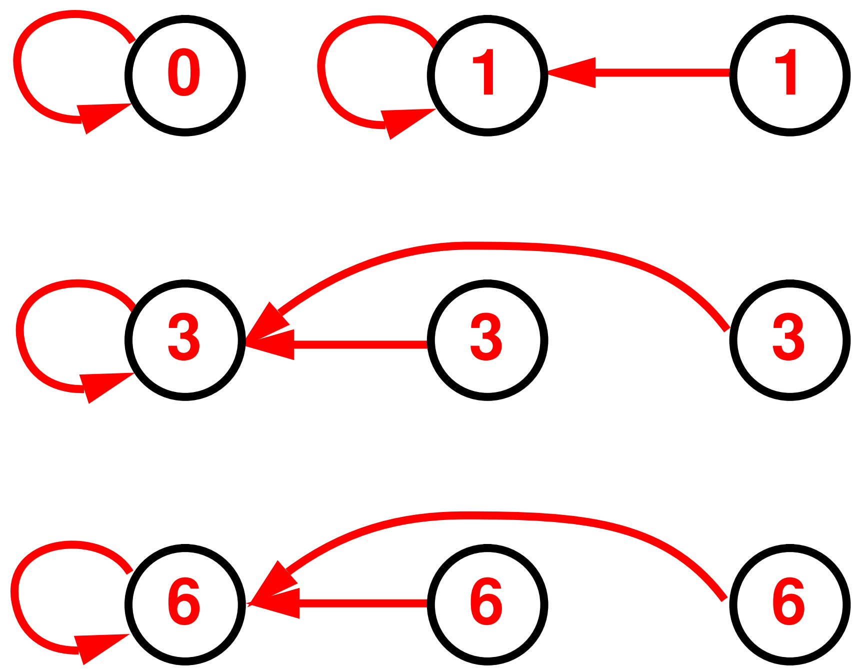
The graph



Initial trees

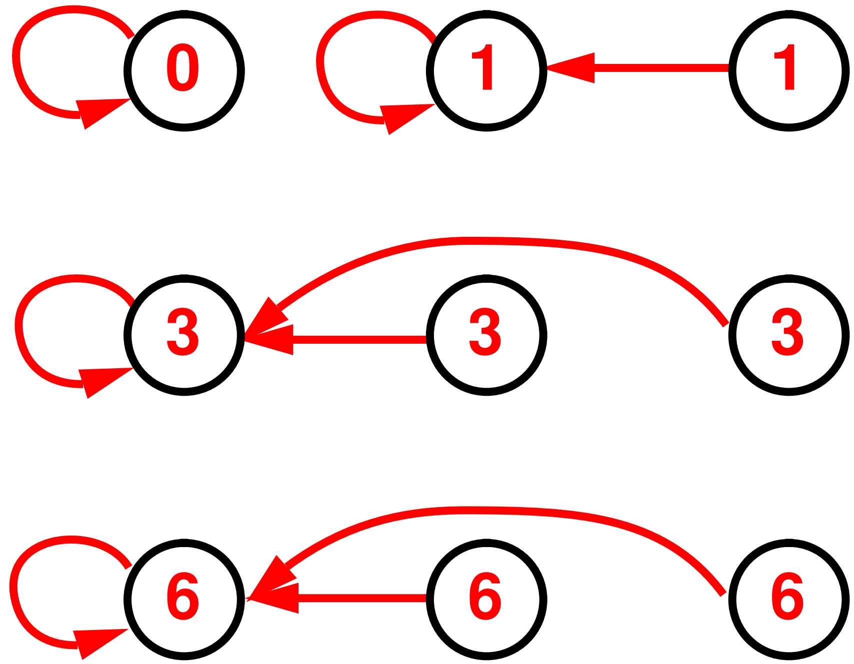


Trees after hooking

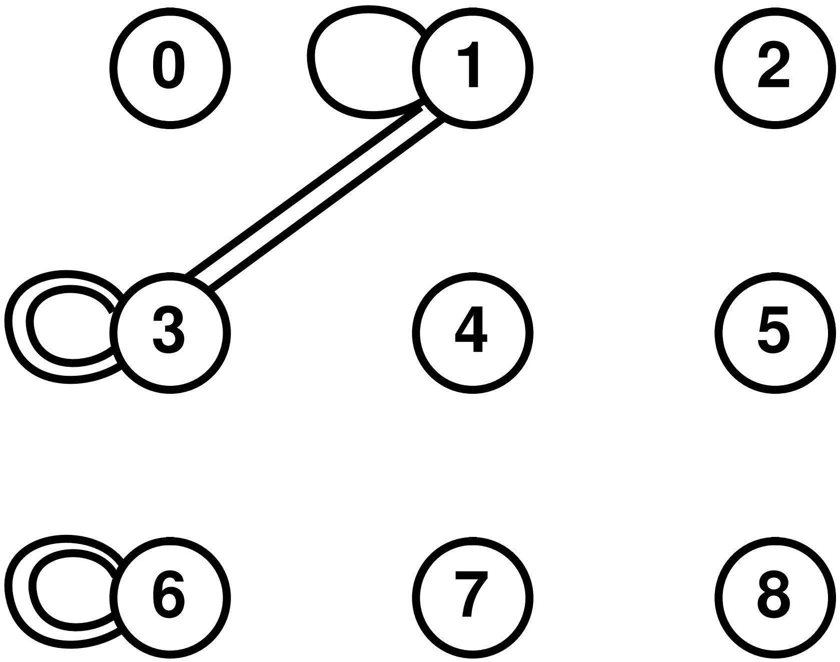


Trees after shortcuting

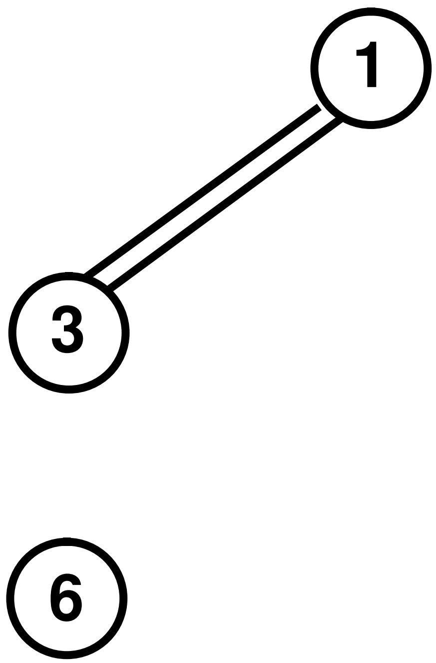
Hybrid Algorithm



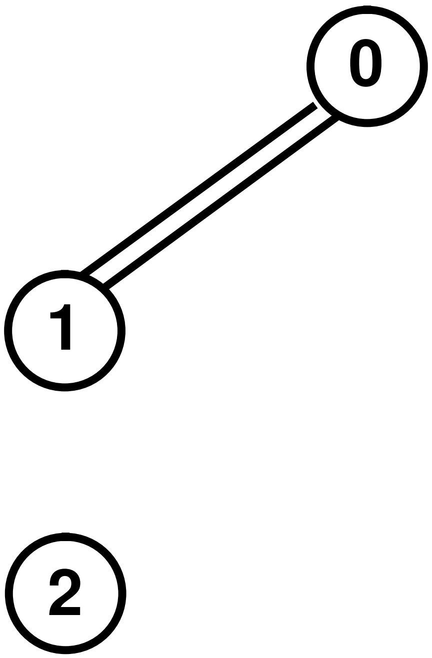
Trees after shortcuting



Graph after renaming edges with parents



Graph after contracting



Graph after renumbering

Hybrid Algorithm in Theory and Practice

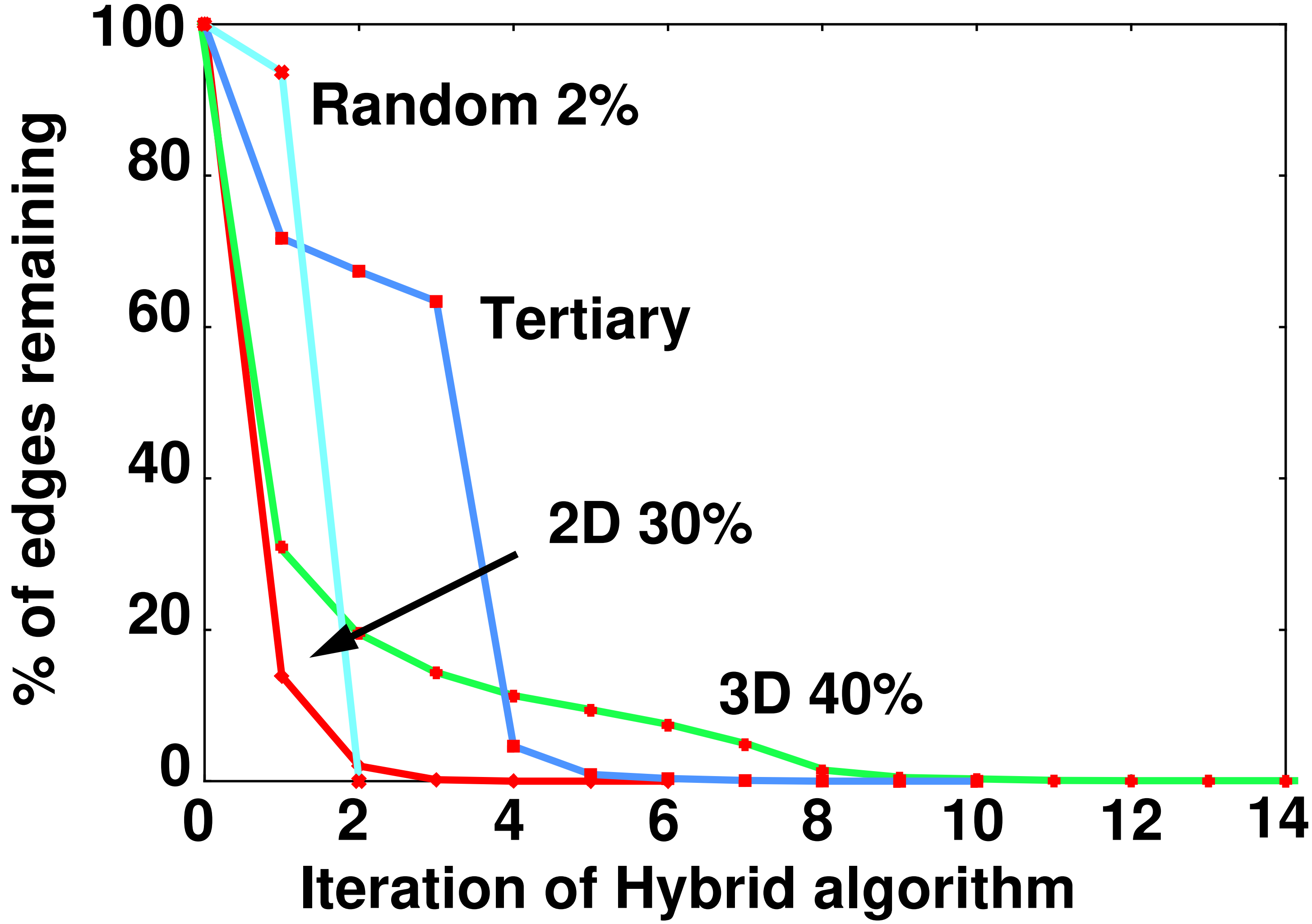
Theory:

Non-optimal, O((m+n) log n) work, O($\log^2$ n) time
O(log n) iterations each of O(m+n) work, O(log n) time

Practice:

Low constants
Simple hooking step
Extensive shortcutting benefits hooking and contraction
Quick contraction — typically removes about 80% of nodes each iteration
Work-efficient for planar graphs — contraction

Rate of Edge Contraction



Concluding Remarks

Hybrid algorithm empirically fastest

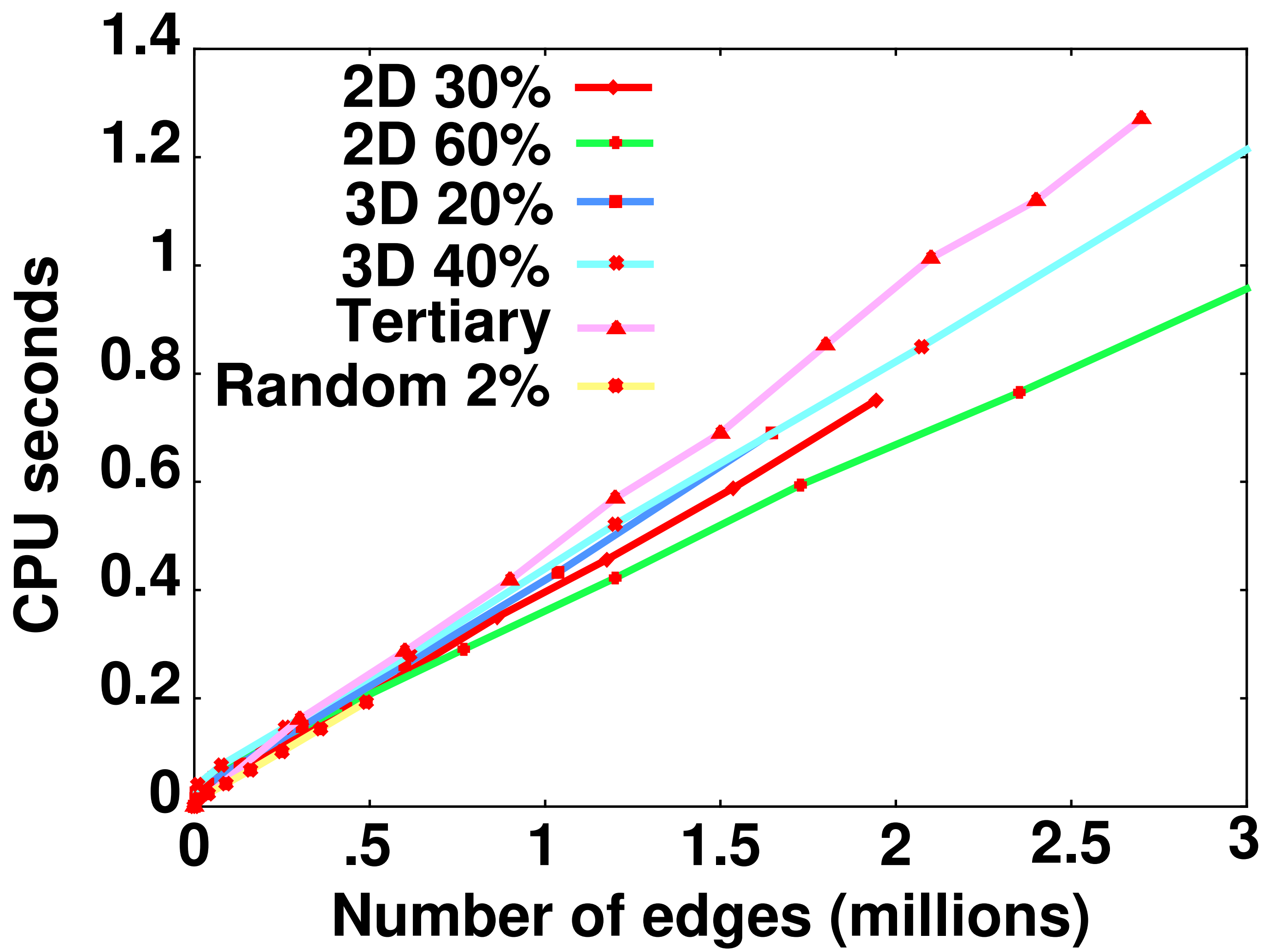
Combines graph contraction and extensive shortcutting
Low constants
Emphasizes many simple steps, rather than few complicated steps
As fast as previous algorithms specialized for grids

On Cray, grids of 2-4M nodes processed in about 1 second
3.5-10.8 times faster than serial DFS on 133 MHz DEC Alpha

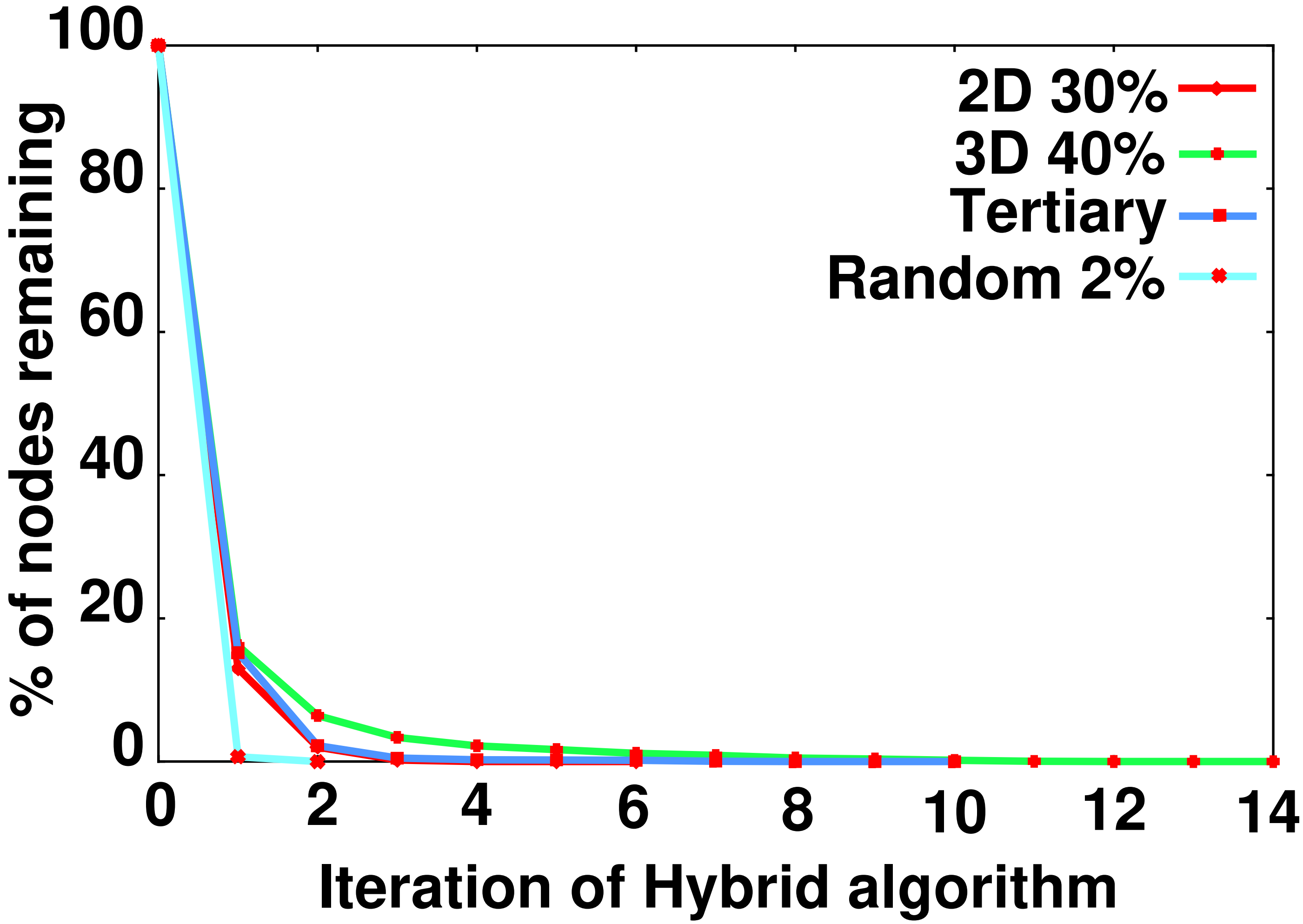
Comparative results very similar on distinct hardware

Results should generalize

Hybrid Algorithm on Cray Y-MP/C90 (1 Proc.)



Rate of Node Contraction



Some details about the classes of graphs

Class	Avg. Degree	Separator Size	Diameter
2D: 30%, 60%	1.2, 2.4	$O(n^{1/2})$	$O(n^{1/2})$
3D: 20%, 40%	1.2, 2.4	$O(n^{2/3})$	$O(n^{1/3})$
Tertiary	6	$O(n)$	$O(\log n)$
Random: 2%	.04 (n-1)	$O(n^2)$	$O(.02\, n^2)$

Hybrid code

```
function flip_edges(edges,flips) = {(select(flip,to,from),select(flip,from,to)) : (from,to) in edges; flip in flips};
function flip_up(edges) = flip_edges(edges,{from > to : (from,to) in edges});
function flip_down(edges) = flip_edges(edges,{from < to : (from,to) in edges});
function flip_all(edges) = {(to,from) : (from,to) in edges};

function parents_edges(parents,edges) =
{(pfrom,pto) : pfrom in parents -> edges_froms(edges); pto in parents -> edges_tos(edges)};

function shrink_edges(parents,edges) =
{(pfrom,pto) : (pfrom,pto) in parents_edges(parents,edges) | pfrom /= pto};

function shortcut(parents) = parents -> parents;

function shortcut_n(parents,n) = if n <= 0 then parents else shortcut_n(shortcut(parents),n-1);

function shortcut_max(parents) =
let gparents = shortcut_n(parents,3); ggparents = shortcut(gparents);
in if all({gp = ggp : gp in gparents; ggp in ggparents}) then ggparents else shortcut_max(ggparents);
```

Hybrid code

```
function compress(newnodes,oldnodes,edges,iter) =
let flags = {oldn == newn and -1 /= c: oldn in oldnodes; newn in newnodes;
                                        c in (dist(-1,#oldnodes) <- edges) <- flip_all(edges)};
   flip_es = if (iter and 1) == 0 then flip_down(edges) else flip_up(edges);
   edges1 = parents_edges(enumerate(flags),flip_es);
in (pack_index(flags),edges1);

function hybrid_alg(nodes,edges,iter) =
if zerop(#edges) then nodes
else let nodes1 = shortcut_max(nodes <- edges);
         edges1 = shrink_edges(nodes1,edges);
         (newnodes,newedges) = compress(nodes1,nodes,edges1,iter);
         roots = hybrid_alg(index(#newnodes),newedges,iter+1);
     in shortcut(nodes1 <- {(i,v) : i in newnodes; v in newnodes -> roots});

function cc_hybrid(edges,numnodes) = hybrid_alg(index(numnodes),flip_up(edges),0);
```