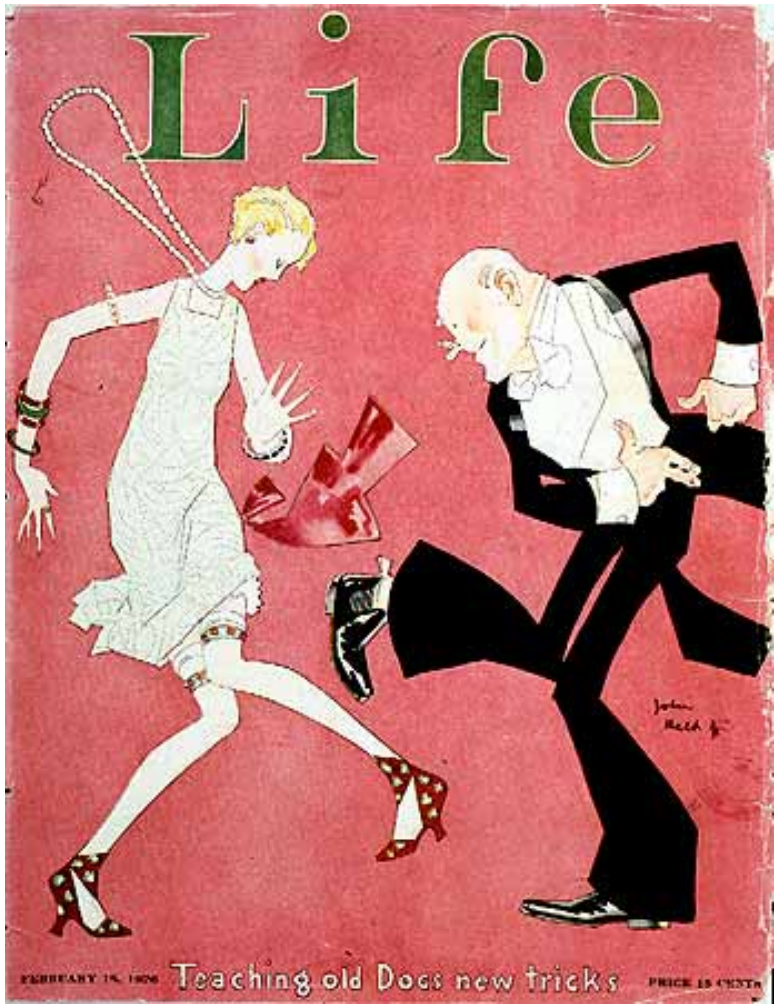
New Tricks for an Old Dog: Improved Max-Flow Algorithms

Andrew V. Goldberg
Microsoft Research – Silicon Valley
www.research.microsoft.com/∼goldberg/



Cartoon by John Held, Jr., Life magazine, 1926.

Problem Definition

- Input: Digraph $G = (V, A)$, $s, t \in V$, $u : A \rightarrow [1, \ldots, U]$.
- $n = |V|$ and $m = |A|$.
- Similarity assumption [Gabow 85]: $\log U = O(\log n)$
  For modern machines $\log U$, $\log n \leq 64$.
- The $\tilde{O}()$ bound ignores constants, $\log n$, $\log U$.
- Flow $f : A \rightarrow [0, \ldots U]$ obeys capacity constraints and conservation constraints.
- Flow value $|f|$ is the total flow into t.
- Cut is a partitioning $V = S \cup T : s \in S, t \in T$.
- Cut capacity $u(S, T) = \sum_{v \in S, w \in T} u(v, w)$.

Maximum flow problem: Find a maximum flow.
Minimum cut problem (dual): Find a minimum cut.

Outline

- Some history.
- Classical algorithms.
- Length functions and improved bounds.
- Push-relabel method.
- Practical implementations.
- Improved implementation.
- Experimental results.

Time Bounds

year	discoverer(s)	bound	note
1951	Dantzig	$O(n^2mU)$	$\tilde{O}\left(n^2mU\right)$
1955	Ford & Fulkerson	$O(m^2U)$	$\tilde{O}\left(m^2U\right)$
1970	Dinitz	$O(n^2m)$	$\tilde{O}\left(n^2m\right)$
1972	Edmonds & Karp	$O(m^2 \log U)$	$\tilde{O}\left(m^2\right)$
1973	Dinitz	$O(nm \log U)$	$\tilde{O}\left(nm\right)$
1974	Karzanov	$O(n^3)$	
1977	Cherkassky	$O(n^2m^{1/2})$	
1980	Galil & Naamad	$O(nm \log^2 n)$	
1983	Sleator & Tarjan	$O(nm \log n)$	
1986	Goldberg & Tarjan	$O(nm \log(n^2/m))$	
1987	Ahuja & Orlin	$O(nm + n^2 \log U)$	
1987	Ahuja et al.	$O(nm \log(n\sqrt{\log U}/m))$	
1989	Cheriyan & Hagerup	$E(nm + n^2 \log^2 n)$	
1990	Cheriyan et al.	$O(n^3/\log n)$	
1990	Alon	$O(nm + n^{8/3} \log n)$	
1992	King et al.	$O(nm + n^{2+\epsilon})$	
1993	Phillips & Westbrook	$O(nm(\log_{m/n} n + \log^{2+\epsilon} n))$	
1994	King et al.	$O(nm \log_{m/(n \log n)} n)$	
1997	Goldberg & Rao	$O(m^{3/2} \log(n^2/m) \log U)$ $O(n^{2/3}m \log(n^2/m) \log U)$	$\tilde{O}\left(m^{3/2}\right)$ $\tilde{O}\left(n^{2/3}m\right)$

blocking flow and push-relabel algorithms.

Augmenting Path Algorithm

- Residual capacity $u_f(a)$ is $u(a) - f(a)$ if $a \in A$ and $f(a^R)$ if $a \notin A$.
- Residual graph $G_f = (V, A_f)$ is induced by arcs with positive residual capacity.
- Augmenting path is an s-t path in G_f.

f is optimal iff there is no augmenting path.

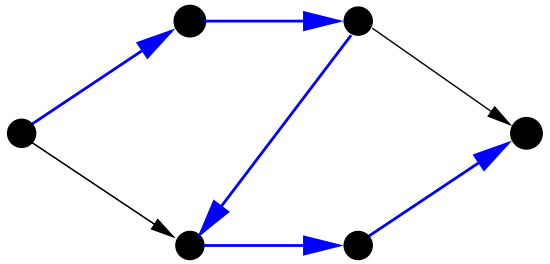
Flow augmentation: Given an augmenting path Γ, increase f on all arcs on Γ by the minimum residual capacity of arcs on Γ.
Saturates at least one arc on Γ.

Augmenting path algorithm: While there is an augmenting path, find one and augment.
Runs in $O(m^2U)$ time.

Unit lengths: $\forall a \in A_f$ let $\ell(a) = 1$.
Augmenting along a shortest path yields a polynomial-time algorithm.

Blocking Flows

f in G is blocking if every s-t path in G is saturated.



- The admissible graph $\overline{G}$ contains all arcs of G_f on s-t shortest paths.
- $\overline{G}$ is acyclic.
- $O(m \log(n^2/m))$ algorithm to find a blocking flow in an acyclic graph [Goldberg & Tarjan 90].

Blocking flow method:[Dinitz 70]
Repeatedly augment f by a blocking flow in G_f.

Lemma: Each iteration increases the s to t distance in G_f.

$O(nm \log(n^2/m))$ maximum flow algorithm.

Binary Length Function

How does one beat the nm barrier?

Algorithm intuition [Goldberg & Rao 1997]:

- Capacity-based lengths:
  $\ell(a) = 1$ if $0 < u_f(a) \leq \Delta$, $\ell(a) = 0$ otherwise.
- Maintain residual flow bound F, set $\Delta = F/\sqrt{m}$.
- Contract SCCs induced by zero-length (fat) arcs.
- This makes $\overline{G}$ acyclic.
- Find a flow of value Δ or a blocking flow; augment.
- $O(\sqrt{m})$ Δ-augmentations.
- After $2\sqrt{m}$ blocking flow augmentations, $d(s) \geq 2\sqrt{m}$. Have a cut with at most $\sqrt{m}/2$ length one arcs.
  Decrease F by a factor of two.

Careful analysis gives a bound of $O(\min(n^{2/3}, \sqrt{m}) \log U)$ blocking flows.

A Confession

The presentation cheats in algorithm description and analysis

- Many important details ignored.
- Example: special arcs in the binary. blocking flow algorithm.
- Data structures not discussed.
- Examples: dynamic tree (or where log's come from).

Results are deep, but the intuition is simple.

Push–Relabel Method

Push–relabel algorithms [Goldberg & Tarjan 1986] are more practical than blocking flow algorithms. Not preflow–push!

- Preflow f [Karzanov 1974]: $v \neq s$ may have flow excess $e_f(v)$, but not deficit.
- Distance labeling gives lower bounds on distance to t in G_f. Formally $d : V \rightarrow \mathcal{N}$, $d(t) = 0$, $\forall (v, w) \in G_f$, $d(v) \leq d(w) + 1$.
- Initially $d(v) = 1$ for $v \neq s, t$, $d(s) = n$, arcs out of s are saturated.
- Apply push and relabel operations until none applies.
- Algorithm terminates with a min-cut. Converting preflow into flow is fast.
- Admissible arc: $(v, w) \in A_f : d(v) > d(w)$.

Push–Relabel (cont.)

- Algorithm updates f and d using push and relabel operations.
- push(v, w): $e_f(v) > 0$, (v, w) admissible.
  Increase $f(v, w)$ by at most $\min(u_f(v, w), e_f(v))$.
- relabel(v): $d(v) < n$, no arc (v, w) is admissible.
  Increase $d(v)$ by 1 or the maximum possible value.
- Current arc data structure: Current arc of v starts at the first arc of v initially and after each relabeling; advances only if the current arc is not admissible.
- Selection rules: Pick the next vertex to process, e.g., FIFO on vertices with excess, highest-labeled vertex with excess.

Efficient Implementations

HI-PR [Cherkassky & Goldberg 1994]

- Layers: Keep vertices v in buckets $d(v)$.
- Highest-level selection.
  Cheap using layers.
- Global update: periodically recompute distances to t.
  Use amortization to control cost.
- Gap: If a layer k is empty, delete vertices in higher layers.
  Cheap using layers.

14 years later:

- The method successfully used in many applications, as a subroutine, extended to parametric and min-cost flows.
- For some applications specialized algorithms do better (e.g. [Boykov & Kolmogorov 2004]).
- Claims of more robust general-purpose algorithms.

We attempt to improve HI-PR (motivated by the workshop).

General Lengths: Practicality

Non-unit lengths are a natural idea with a solid theoretical justification, but...

- [Hagerup 1998]: The binary blocking flow algorithm implementation more robust that the standard one.
- So far, nobody was able to use length functions to get a more robust implementation than HI-PR (we tried!).
- Theoretical obstacle – flow can move around cycles.
- Global recomputation of distances and contraction of the SCCs is expensive.

Augment–Relabel Algorithm

Intuitively, push-relabel with DFS operation ordering.

```
FindPath(v)
{
  if (v == t) return(true);
  while (there is an admissible arc (v,w)) {
    if (FindPath(w) {
      v->current = (v,w); return(true);
    }
  }
  relabel(v); return(false);
}
```

The algorithm repeatedly calls `FindPath(s)` and augments along the current arc path from s to t until $d(s) \geq n$.
Can use binary lengths to get the improved bounds.
Does not work well in practice.

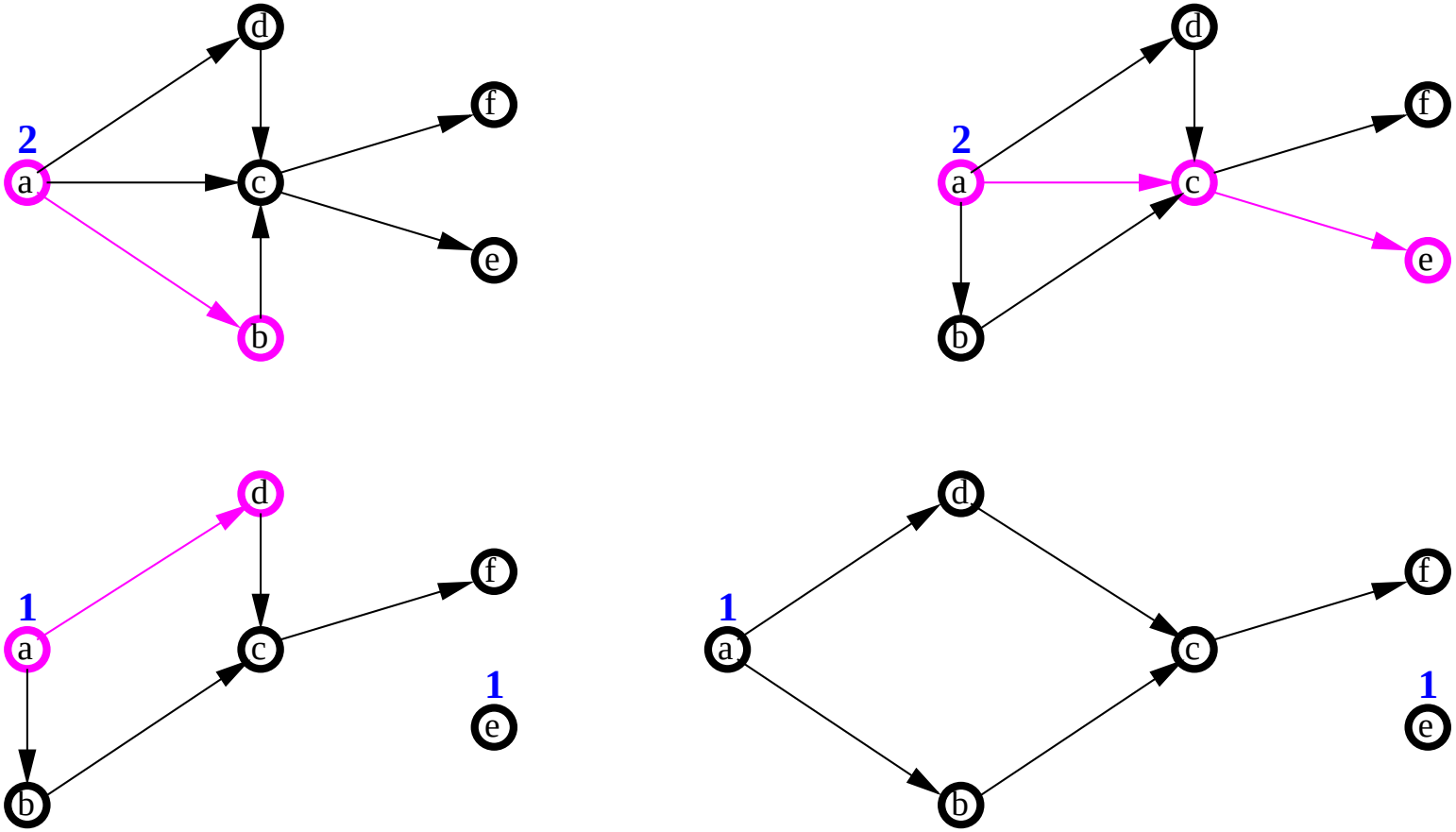
Partial Augment–Relabel (PAR)

PAR-k algorithm picks $v : 0 < d(v) < n, e_f(v) > 0$ and tries to find a length-k admissible path from v or relabel v.

```
FindPath(v,k)
{
  if ((k == 0) or (v == t)) return(true);
  while (there is an admissible arc (v,w)) {
    if (FindPath(w, k-1) {
      v->current = (v,w); return(true);
    }
  }
  relabel(v); return(false);
}
```

If a path is found, flow is pushed along the path (as a sequence of push operations).

PAR-2 Example



Unit residual capacities, $k = 2$, layered by distances.

PAR-k Implementation

Many similarities to HI-PR: Use current arc data structure, gap heuristic, layers, (improved) global updates.

- Use the unary length function.
- Highest-level selection.
- $k = 4$ seems to be a good choice.
- Improved global update
  - If layers L and below have not been touched since the last update, start update at layer L.
  - Once all vertices with v with $d(v) < n$ and $e_f(v) > 0$ have been reached, stop the update.
  - Heavier use of amortization in update scheduling.
- Use fast (+1) relabel.

Experimental Setup

- DIMACS problem families.
- Randomly re-number vertices and sort arcs by tail IDs.
- Go to the biggest problem size that fits in RAM.
- HP Evo D530 machine, 2Gig RAM, 3.2 Ghz processor, Fedora 7 Linux, `gcc/g++` compilers with `-O4` optimization.
- Time in seconds, averaged over 10 instances.
- Scan count: relabel and global update scans.
- Compare PAR to HI-PR 3.6; also to [Chandran & Hochbaum] codes CH-3.1, CH-3.21.

RMF Families

RMF-LONG: Assymptotic improvement to almost linear time.

m n	1.3 M 0.3 M	2.6 M 0.5 M	5.1 M 1.1 M	10.2 M 2.0 M	20.3 M 4.1 M
HI-PR tm	0.67	1.75	4.26	12.63	41.88
PAR tm	0.41	0.82	1.71	3.60	7.53
HI-PR sc/n	8.16	11.25	13.25	20.04	33.97
PAR sc/n	5.20	5.16	5.32	5.69	5.71

RMF-WIDE: Small improvement. (PAR scans are cheaper.) Superlinear time, hardest problems tested.

m n	0.6 M 0.1 M	1.3 M 0.3 M	2.6 M 0.5 M	5.2 M 1.0 M	10.3 M 2.1 M	20.7 M 4.2 M
HI-PR tm	2.39	6.45	15.05	41.55	100.12	266.99
PAR tm	1.97	5.35	12.81	31.06	74.93	184.95
HI-PR sc/n	41.39	52.93	60.90	76.55	88.64	102.43
PAR sc/n	53.79	68.82	77.75	93.80	112.37	132.04

Moderate (1M arcs) porblems solved in seconds, large (20 M) – in seconds or minutes.

Washington Families

WASH-LONG: Small improvement. Problems get simpler as n grows, large ones trivial.

m n	1.6 M 0.5 M	3.1 M 1.0 M	6.3 M 2.1 M	12.6 M 4.2 M	25.2 M 8.4 M
HI-PR tm PAR tm	0.82 0.57	1.60 1.03	3.05 1.89	5.59 3.62	10.82 6.97
HI-PR sc/n PAR sc/n	3.98 4.14	3.62 3.12	3.26 2.56	2.68 2.28	2.55 2.08

WASH-WIDE: Small improvement.

m n	0.8 M 0.18 M	1.6 M 0.37 M	3.1 M 0.53 M	6.3 M 1.0 M	12.5 M 2.1 M	25.0 M 4.2 M
HI-PR tm PAR tm	2.36 1.37	7.61 4.19	22.10 12.30	51.81 29.09	133.37 74.55	273.51 153.58
HI-PR sc/n PAR sc/n	19.99 16.91	25.51 20.95	30.46 26.97	32.32 29.08	39.92 37.07	39.22 36.01

Other Families

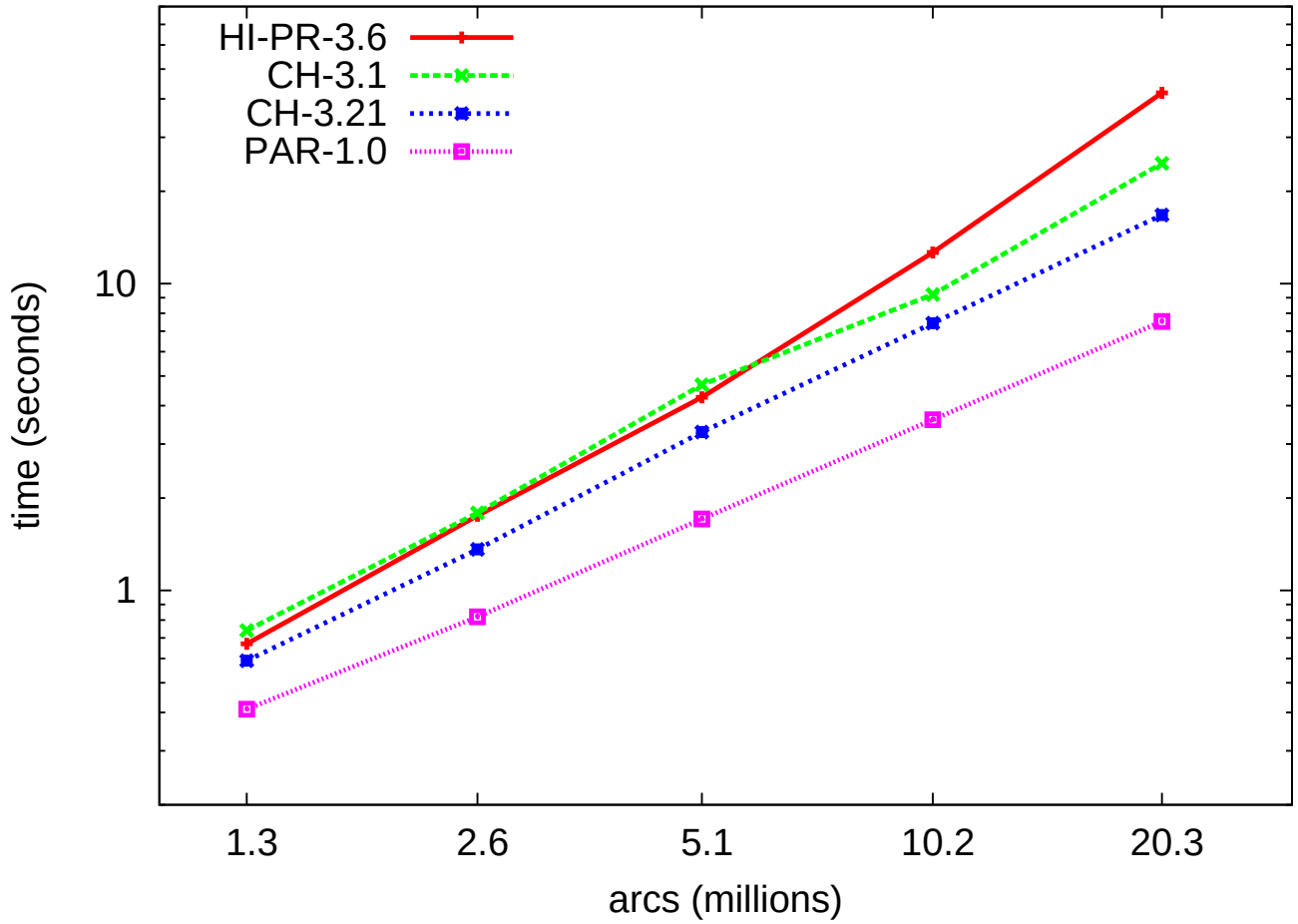
WASH-MODERATE: Trivial problems.

m n	2.1 M 41 K	4.2 M 66 K	8.4 M 104 K	16.8 M 165 K	33.5 M 262 K
HI-PR tm	0.24	0.46	1.06	2.26	4.50
PAR tm	0.16	0.29	0.67	1.49	3.05
HI-PR sc/n	2.07	2.06	2.05	2.05	2.04
PAR sc/n	1.13	1.11	1.10	1.09	1.07

ACYC-DENSE: Improvement big, but problems problems are trivial. Performance very sensitive to initialization.

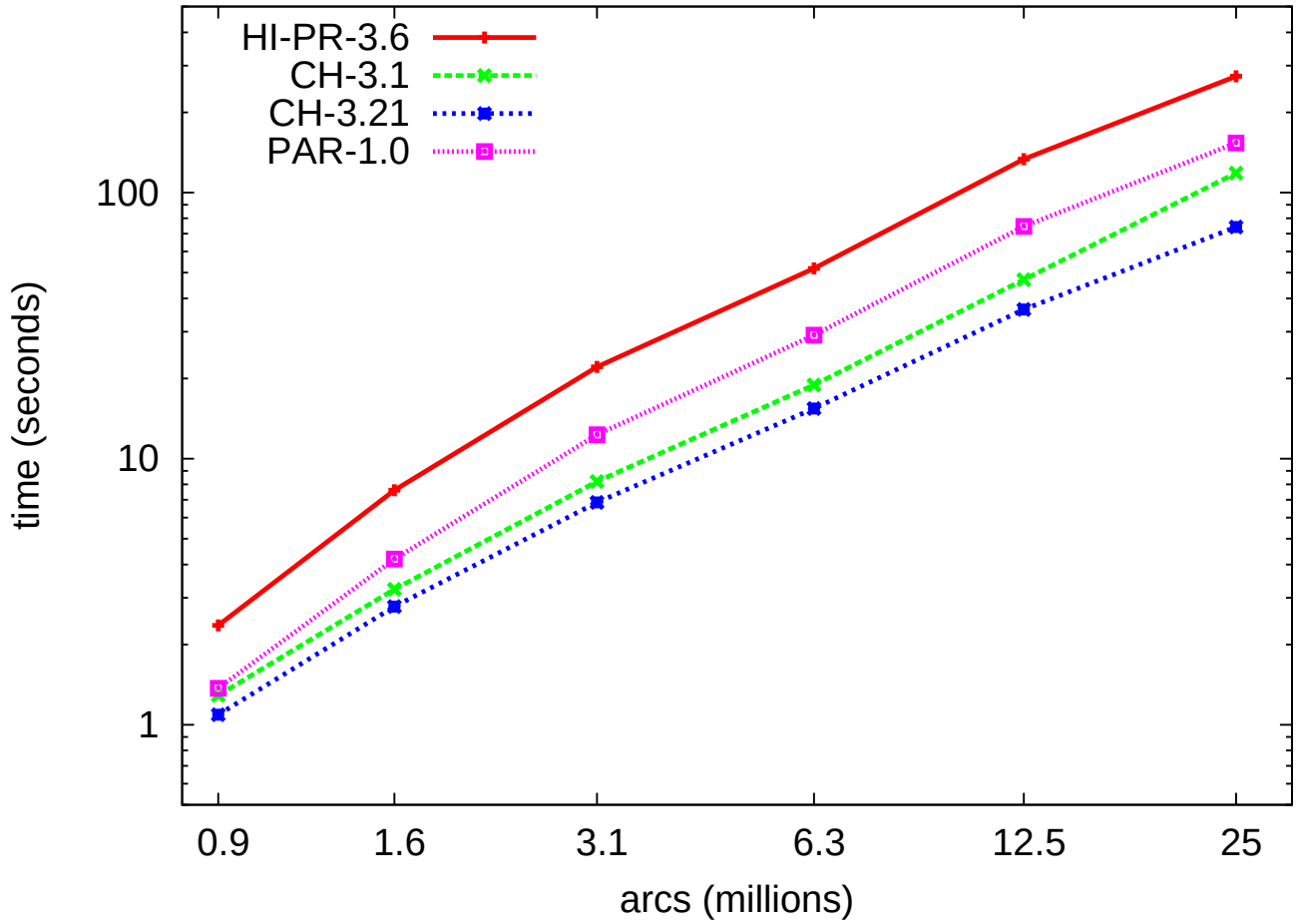
m n	2.1 M 2.0 K	4.2 M 2.9 K	8.4 4.1 K	16.8 M 5.8 K	33.6 M 8.2 K
HI-PR tm	0.93	1.93	4.60	10.63	27.08
PAR tm	0.11	0.22	0.46	0.90	1.80
HI-PR sc/n	8.81	9.12	10.50	11.25	12.39
PAR sc/n	2.51	2.46	2.68	2.65	2.68

RMF-LONG Family



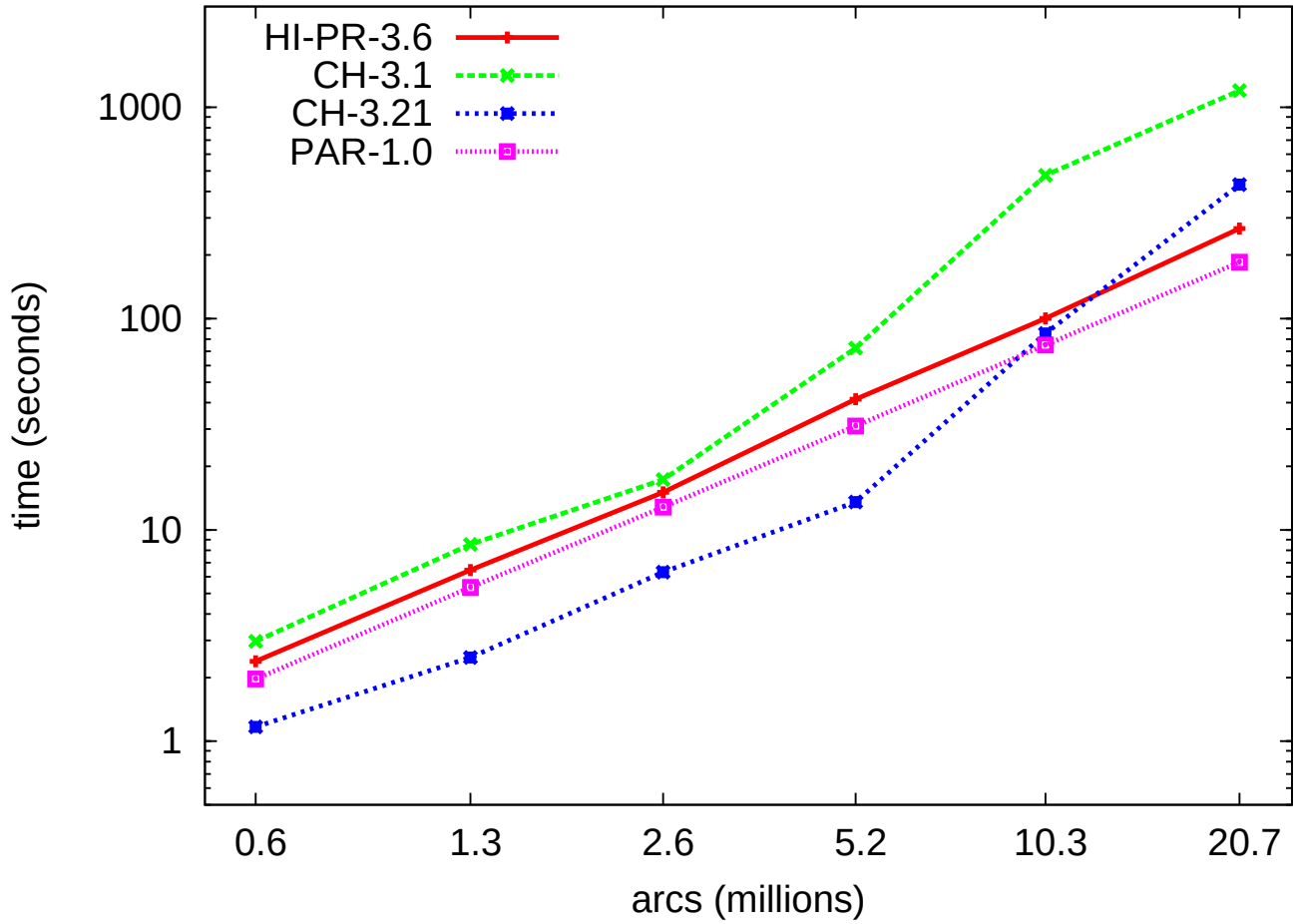
PAR assymptotically fastest.

WASH-WIDE Family



CH codes are faster; PAR is within a factor of ≈ 2 of CH-3.21.

RMF-WIDE Family



CH codes have high variance, assymptotically slower.

Concluding Remarks

- DIMACS testbed needs to be expanded.
  Interesting instances from vision applications?
  Are real-life problems ever hard?
- Application-specific improvements:
  - Locality-aware graph layout.
  - Compact graph representation.
  - Good initial solution.
  - Incremental applications.
- PAR improves on HI-PR.
- CH codes appear less robust than PAR.
- Further improvements to the push-relabel method.
- Binary and other length functions.
- Implications to related algorithms, such as global minimum cut, parametric flow, minimum-cost flow.

Thank You!