The Binary Blocking Flow Algorithm

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

Theory vs. Practice

In theory, there is no difference between theory and practice.

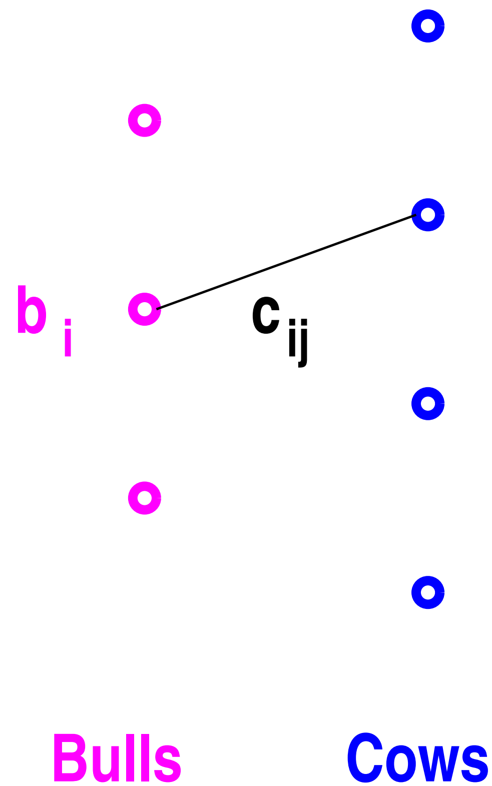
Problem Definition

- 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.

Applications of Flows

- Classical OR applications, e.g., open pit mining, logistics.
- Recent applications in computer vision, e.g., image segmentation and stereo vision.
- Recent web applications like document classification.
- AI application.



Artificial Insemination.

Outline

- History.
- The blocking flow method.
- The binary blocking flow algorithm.
- Open problem: making the algorithm practical.
- Open problem: extending the result to minimum-cost flows.

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.

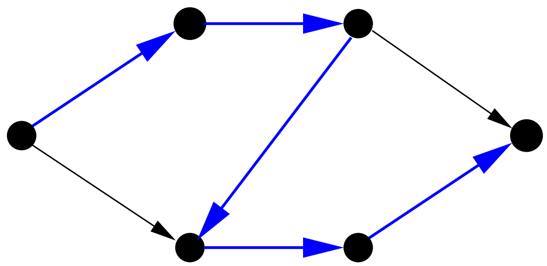
Flow augmentation: Given an augmenting path Γ, increase f on all arcs on Γ by the minimum residual capacity of arcs on Γ.
Saturates at least one arc on Γ.

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.

Blocking Flows: Analysis

Main lemma: Each iteration increases the s to t distance in G_f.
Proof: Let d be the shortest path distance function (to t).
Augmentation changes $\overline{G}$.

- Saturated arcs deleted, distances do not decrease.
- For new arcs (v, w), $d(v) < d(w)$, distances do not decrease.
- For the new $\overline{G}$ and old d, every s-t path contains an arc (v, w) with $d(v) \leq d(w)$ by the definition of the blocking flow.
- The s-t distance increases.

Theorem: The blocking flow algorithm can be implemented to run in $O(nm \log(n^2/m))$ time.

Decomposition Barrier

- A flow can be decomposed into $O(m)$ paths of length $O(n)$.
- The total length of augmenting paths can be $\Omega(nm)$.
- Without data structures, the blocking flow algorithm takes $\Omega(nm)$ time.
- But data structures allow changing flow on many arcs in one operation.

Can we beat the $\Omega(nm)$ barrier?

For unit capacities, the blocking flow algorithm runs in $O(\min(m^{1/2}, n^{2/3})$ time [Karzanov 73] [Even & Tarjan 74].

Unit Capacities

Theorem: For unit capacities, the blocking flow algorithm terminates less than $2\sqrt{m}$ iterations.
Proof:

- After $\sqrt{m}$ iterations, $d(s) > \sqrt{m}$.
- Consider cuts $(\{d(v) > i\}, \{d(v) \leq i\})$.
- A residual arc crosses at most one such cut.
- One of the cuts' residual capacity is below $\sqrt{m}$.
- Less than $\sqrt{m}$ additional iterations.

A slightly different argument gives an $O(n^{2/3})$ bound.

Binary Length Function

Algorithm intuition [Goldberg & Rao 1997]:

- Capacity-based lengths:
  $\ell(a) = 1$ if $0 < u_f(a) < 2\Delta$, $\ell(a) = 0$ otherwise.
- Maintain residual flow bound F, update when improves by at least a factor of 2.
- Set $\Delta = F/\sqrt{m}$.
- Find a flow of value Δ or a blocking flow; augment.
- After $O(\sqrt{m})$ Δ-augmentations F decreases.
- After $4\sqrt{m}$ blocking flow augmentations, $d(s) \geq 2\sqrt{m}$.
- One of the cuts $(\{d(v) > i\}, \{d(v) \leq i\})$ has no 0-length arcs and at most $\sqrt{m}/4$ length one arcs.
- After $O(\sqrt{m})$ blocking flows F decreases.

Why stop blocking flow computation at Δ value?

Zero Length Arcs

Pros:

- Seem necessary for the result to work.
- Large arcs do not go from high to low vertex layers.
- Small cut when $d(s) << n$.

Cons:

- $\overline{G}$ need not be acyclic.
- Increasing flow in $\overline{G}$ may create new admissible arcs: $d(v) = d(w)$, increasing $f(v, w)$ may increases $u_f(w, v)$ to 2Δ.
- The new arcs are created only if an arc length is reduced to zero.

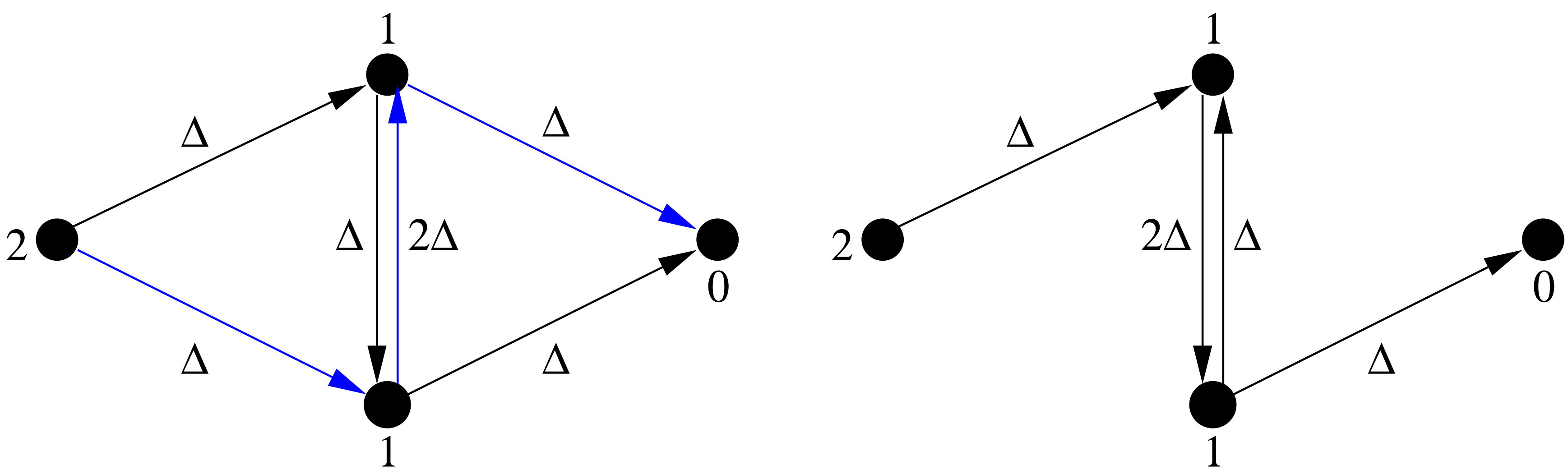
These problems can be resolved.

Problem: Admissible Cycles

The admissible graph $\overline{G}$ is induced by arcs $(v,w) \in G_f : d(v) = d(w) + \ell(v,w)$.

- $\overline{G}$ can have only cycles of zero-length arcs between vertices with the same d.
- These arcs have capacities of at least 2Δ.
- Contract SCCs of $\overline{G}$ to obtain acyclic $\overline{G}'$.
- Δ flow can be routed in such a strongly connected graph in linear time [Erlebach & Hagerup 02, Haeupler & Tarjan 07].
- Stop a blocking flow computation if the current flow has value Δ.
- After finding a flow in $\overline{G}'$, extend it to a flow in $\overline{G}$.
- A blocking flow in $\overline{G}'$ is a blocking flow in $\overline{G}$.

Problem: Arc Length Decrease



An arc length can decrease from one to zero and s-t distance may not increase.

Special Arcs

When can length decrease on (v, w) happen and hurt?

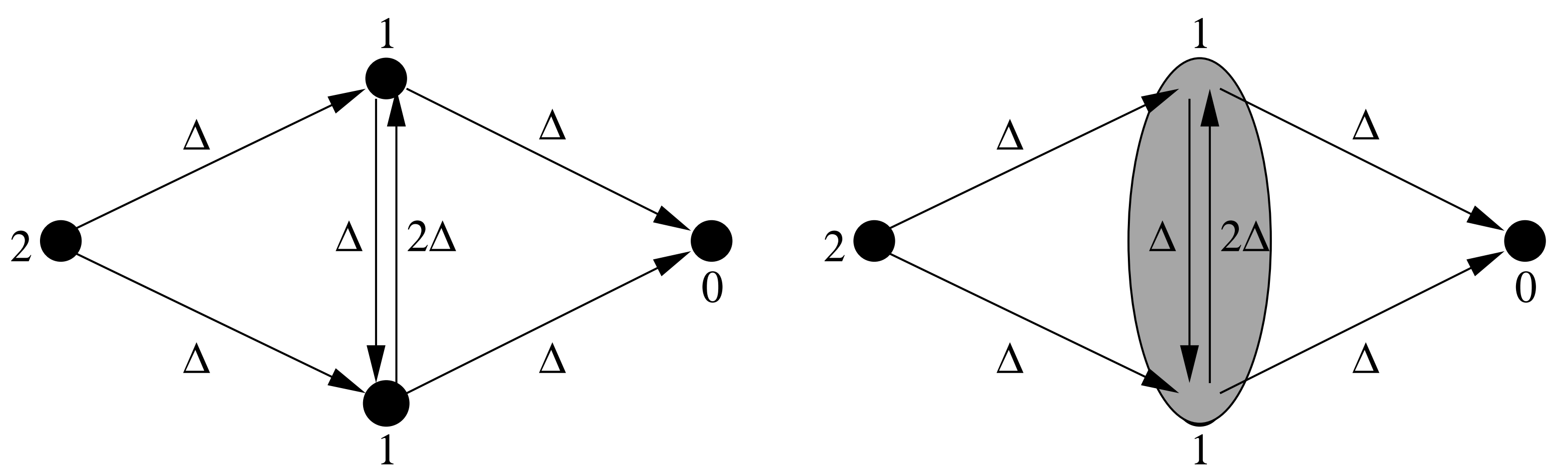
1. $\Delta \leq u_f(v, w) < 2\Delta$
2. $d(v) = d(w)$
   - $d(v) > d(w)$: $f(v, w)^R$ not increases, $\ell(v, w)$ not decreases.
   - $d(v) < d(w)$: decreasing $\ell(v, w)$ does not hurt.
3. (optional) $u_f(v, w)^R \geq 2\Delta$

Special arc: Satisfies (1), (2) and optionally (3).

Can reduce special arc length to zero: d does not change, residual capacity large.

Main Loop

- Assign arc lengths, compute distances to t.
- Reduce special arc length to zero.
- Contract SCCs in $\overline{G}$ to obtain $\overline{G}'$.
- Find a Δ-flow or a blocking flow in $\overline{G}'$.
- Extend to a flow in $\overline{G}$, augment.



Main Theorem

Theorem: While F stays the same, d is monotone. In the blocking flow case, $d(s)$ increases.

Proof: Similar to the regular blocking flow algorithm except for special arcs.

Analysis

$O(\sqrt{m}\log(mU))$ iteration bound is easy. To do better:

- While $\Delta \geq U$ no zero-length arcs, $d(s)$ monotone.
- After $O(\sqrt{m})$ iterations $F \leq \sqrt{m}U$.
- $O(\sqrt{m})$ iterations reduces F by a factor of two.
- In $O(\sqrt{m}\log U)$ iterations $F \leq \sqrt{m}$.
- Integral flow, an iteration decreases F.
- $O(\sqrt{m}\log U)$ iterations total.
- An iteration is dominated by a blocking flow.
- A slight variation gives an $O(n^{2/3}\log U)$ iteration bound.

Theorem: The algorithm runs in $O(\min(m^{1/2}, n^{2/3})m\log(U)\log(n^2/m))$ time.

Practicality

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

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

Open problem: Are non-unit length functions practical?

Push–Relabel Method

Push–relabel algorithms [Goldberg & Tarjan 1986] are more practical than blocking flow algorithms. Uses unit lengths.

- 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 $d : V \to \mathcal{N}$, $d(t) = 0$, $\forall(v, w) \in G_f$, $d(v) \leq d(w) + 1$.
- 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.
- Selection rules: Pick the next vertex to process, e.g., FIFO on vertices with excess, highest-labeled vertex with excess.

The binary lengths function does not give improved bounds.

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.

Min-Cost Flows

Problem definition: Additional cost per unit of flow $c(a)$; find maximum flow of minimum cost.

Min-cost flow algorithms:

- For unit lengths, max-flow + cost-scaling = min-cost flow with $\log(nC)$ slowdown, where C is the maximum arc length.
- In particular, get an $O(nm \log(n^2/m) \log(nC))$ algorithm.
- For unit capacities, [Gabow & Tarjan 87] give an $O(\min(n^{2/3}m^{1/2})m \log(nC))$ algorithm.

Open problem: For min-cost flows with integral data, is there an $O(\min(n^{2/3}m^{1/2})m \log(nC) \log U)$ algorithm?
...or a more modest $\tilde{O}\left(n^{1-\epsilon}m\right)$ algorithm for $\epsilon > 0$?

Conclusions

- Bounds for unit and arbitrary integral capacity maximum flows are close.
- Strongly polynomial bounds are still $\omega(nm)$.
- Non-unit length functions are natural and theoretically justified, but not practical yet.
- For minimum-cost flow, bounds for unit and arbitrary integral capacities are far.