Tidal Flow: A Fast and Teachable Maximum Flow Algorithm

Matthew C. Fontaine

Algorithms Live!

A weekly live talk show on algorithms in competitive programming.

Blog: http://algorithms-live.blogspot.com/

Channel: https://www.youtube.com/algorithmslive

Maximum Flow Problems in CP

- Problems are interesting
- High variation of problem ideas (vs spanning tree algorithms, shortest paths, etc)
- Harder problems require faster algorithms

Timeline for Tidal Flow

2012		ACM ICPC WF (invented)
2013	}	Taught at U Central Florida
...		
2016		
2017		IOI Conference suggested
2018		Present

Algorithms in Competitive Programming

- Easy to teach
- Easy to implement
- Efficient in practice

Example: Fenwick Tree

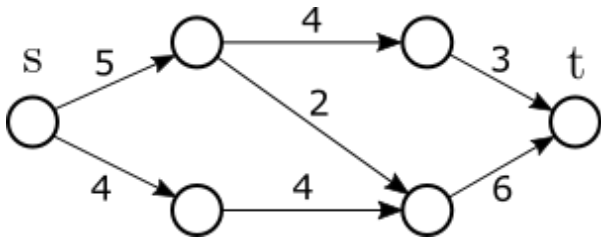
```
void update(int i, int delta) {
   while (i < size) {
      table[i] += delta;
      i += i&-i;
   }
}
```

```
void sum(int i) {
   int sum = 0;
   while (i > 0) {
      sum += table[i];
      i -= i&-i;
   }
   return sum;
}
```

Overview

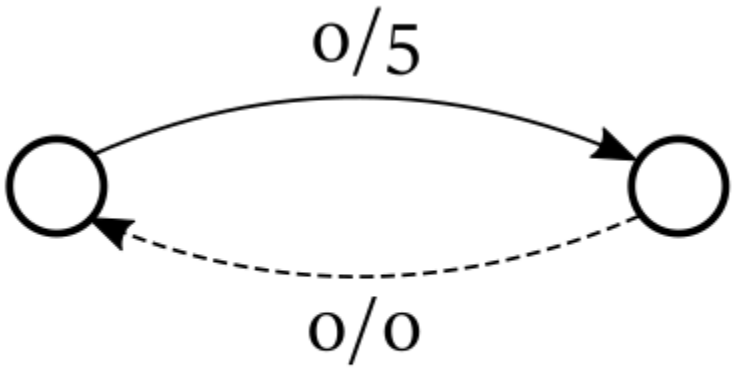
- Review of the maximum flow problem and blocking flows
- Tidal Flow
- Theoretical performance
- Empirical performance
- Open questions

Maximum Flow



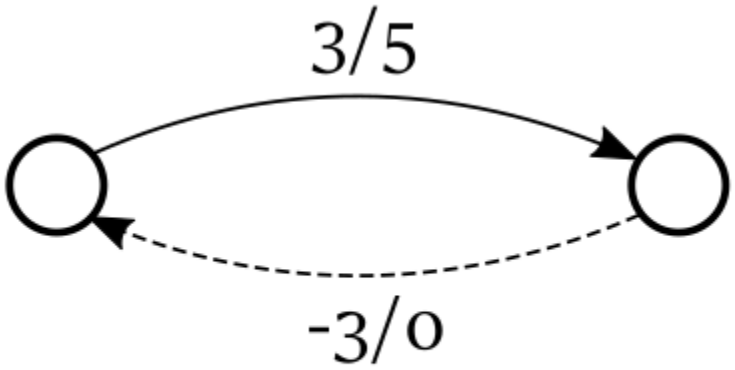
- Input: (G, s, t, cap)
  - $G = (V, E)$
  - $s, t \in V$
  - $cap: E \to \mathbb{R}^+$
- Output: $f: E \to \mathbb{R}^*$
- Constraints:
  - Capacity Constraint: $f(e) \leq cap(e)$ for all $e \in E$
  - Conservation Constraint: $\sum f(w, v) = \sum f(v, u)$ for all $v \in V \setminus \{s, t\}$
- Maximize: $\sum f(s, v)$ for all $v \in V$

Maximum Flow: Reverse Edges



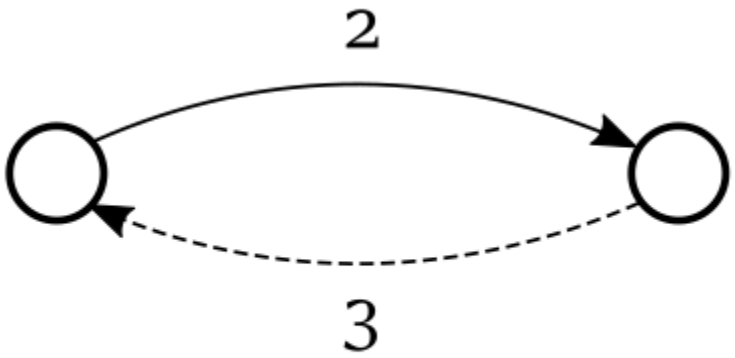
flow/cap

Maximum Flow: Reverse Edges



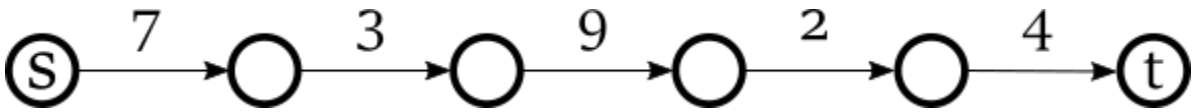
$f(w, v) = -f(v, w)$ for all $(w, v) \in E$

Maximum Flow: Residual Capacities

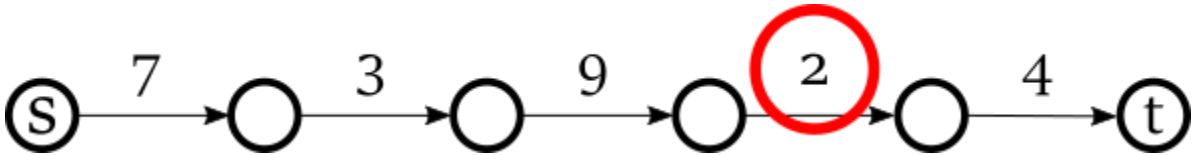


$$cap_r(w, v) = cap(w, v) - f(w, v) \text{ for all } (w, v) \in E$$

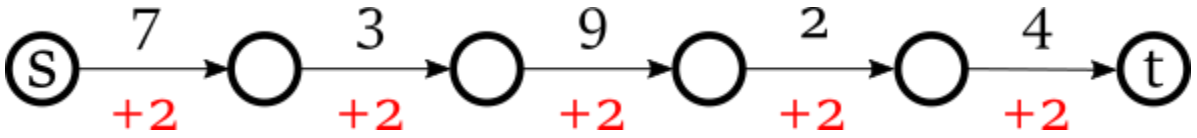
Augmenting Paths



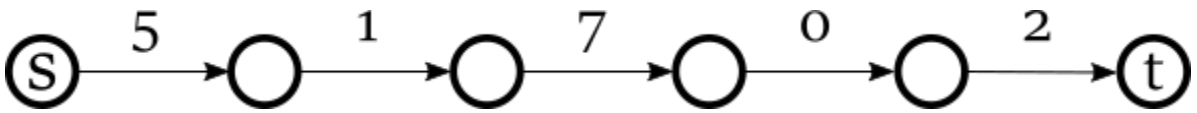
Augmenting Paths



Augmenting Paths



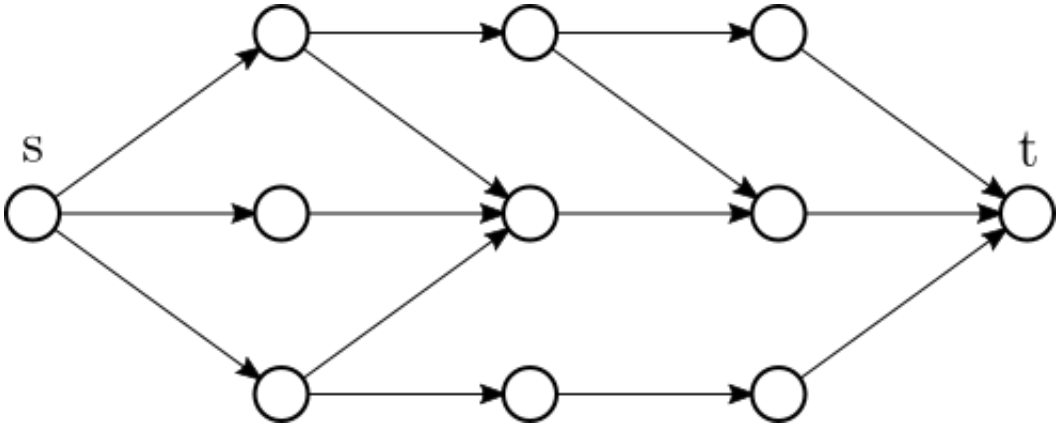
Augmenting Paths



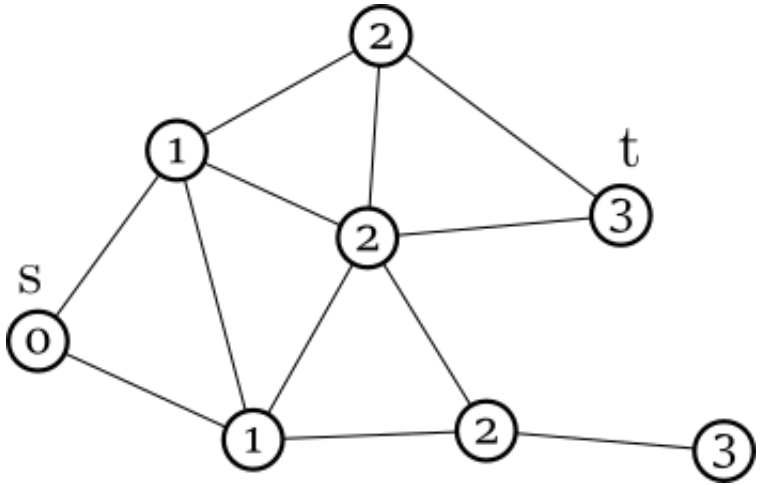
Augmenting Paths: Runtime

- Arbitrary Paths: $O(fm)$
- Shortest Paths: $O(nm^2)$

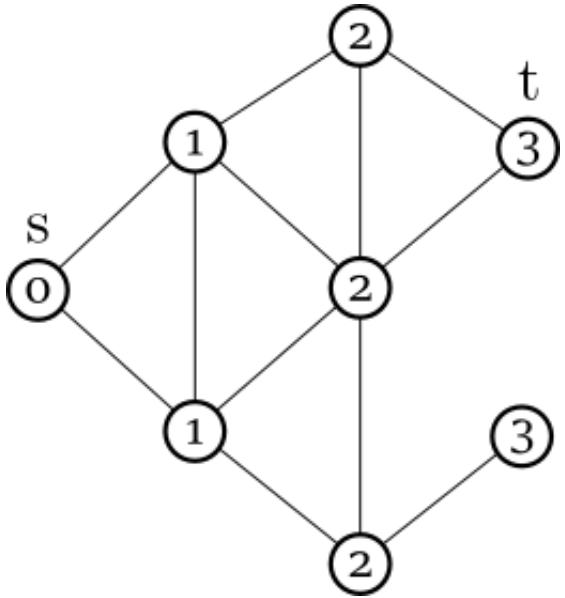
Dinitz Algorithm



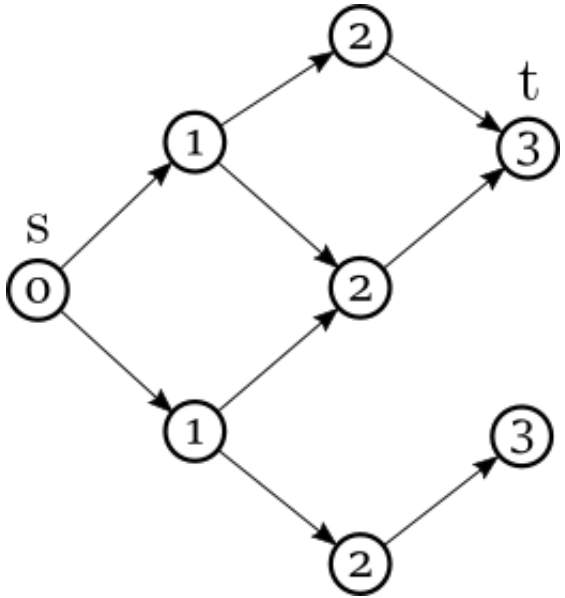
Level Graph



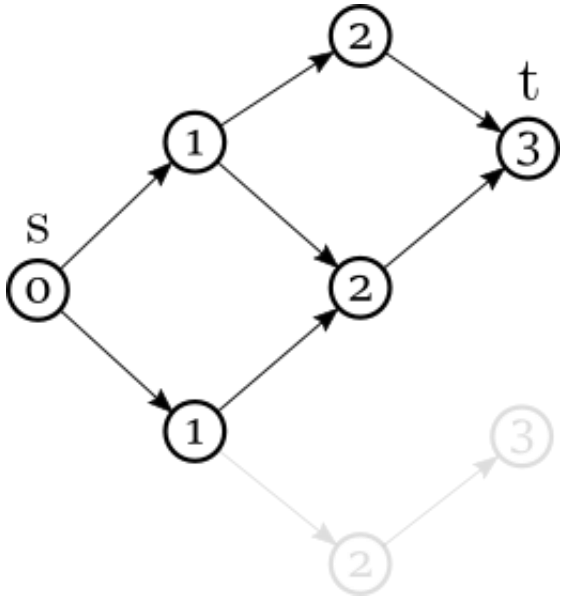
Level Graph



Level Graph



Level Graph



Dinitz

- Blocking flows discovered in $O(nm)$ through modified DFS
- At most $O(n)$ level graphs possible
- Runtime: $O(n^2m)$

Tidal Flow

- Same goal as Dinitz: block the level graph
- Try blocking in $O(m)$ runtime
- Failure is OK! Keep trying.

Tidal Flow: Conceptual Metaphor

“A (conceptual) metaphor is a cognitive process that occurs when a subject seeks understanding of one idea (the target domain) in terms of a different, already known idea (the source domain). The subject creates a conceptual mapping between the properties of the source and the target, thereby gaining new understanding about the target.” (Forišek and Stienová, 2013)

Here the metaphor is oceanic tides.

Tide Cycle

- High Tide
- Low Tide
- Erosion

Tide Phase: High Tide

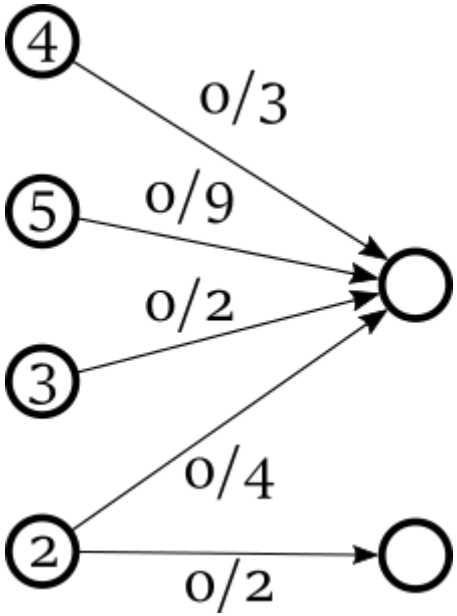
- For each vertex, compute $h: V \to R^*$

$$h(v) = \sum_{w \in G(v)} \min(cap(w,v) - f(w,v), h(w))$$

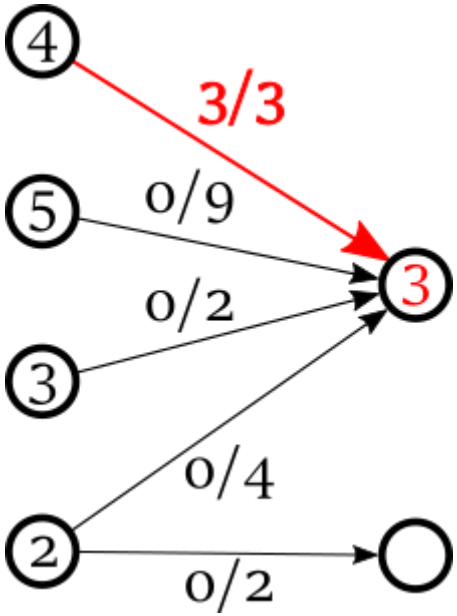
- Store promised flow $p: E \to R^*$

$$p(w,v) = \min(cap(w,v) - f(w,v), h(w))$$

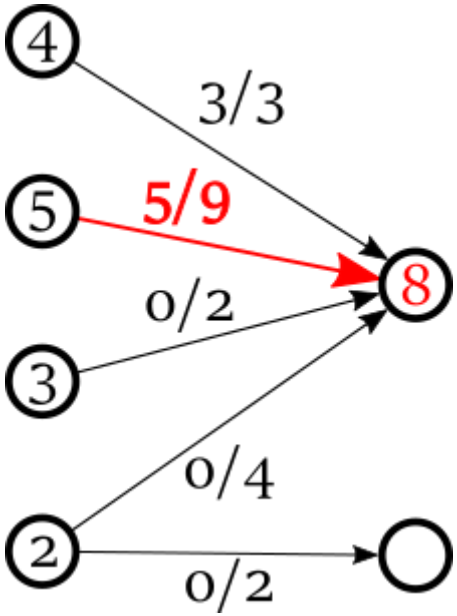
High Tide



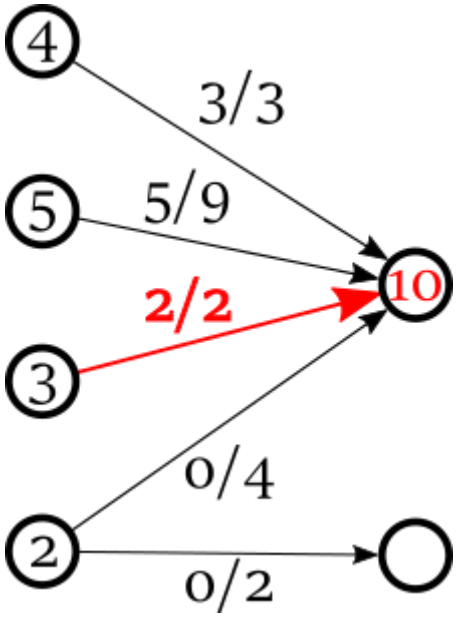
High Tide



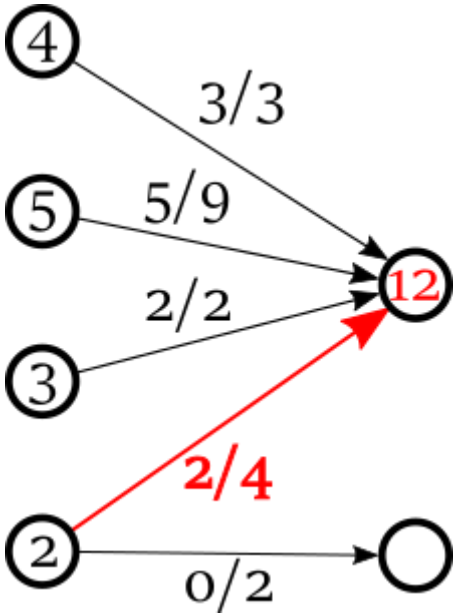
High Tide



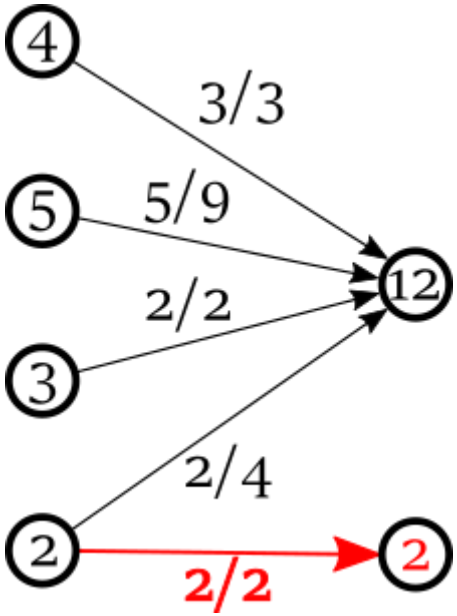
High Tide



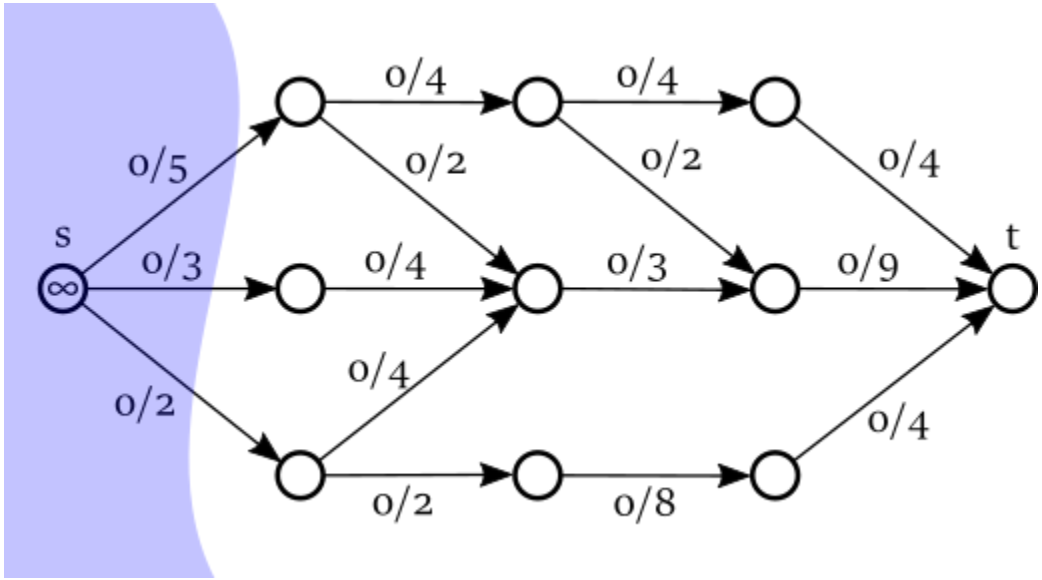
High Tide



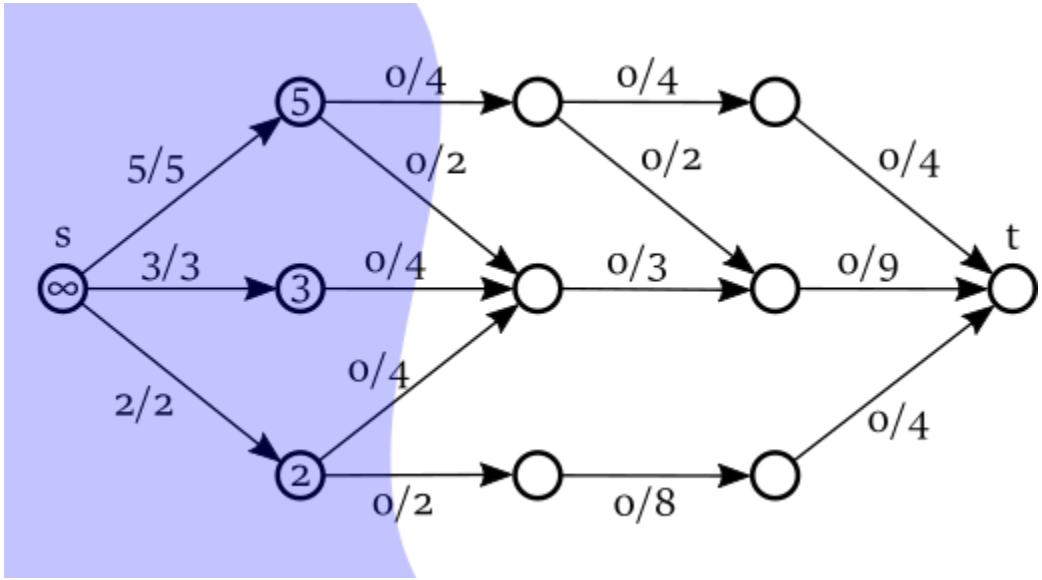
High Tide



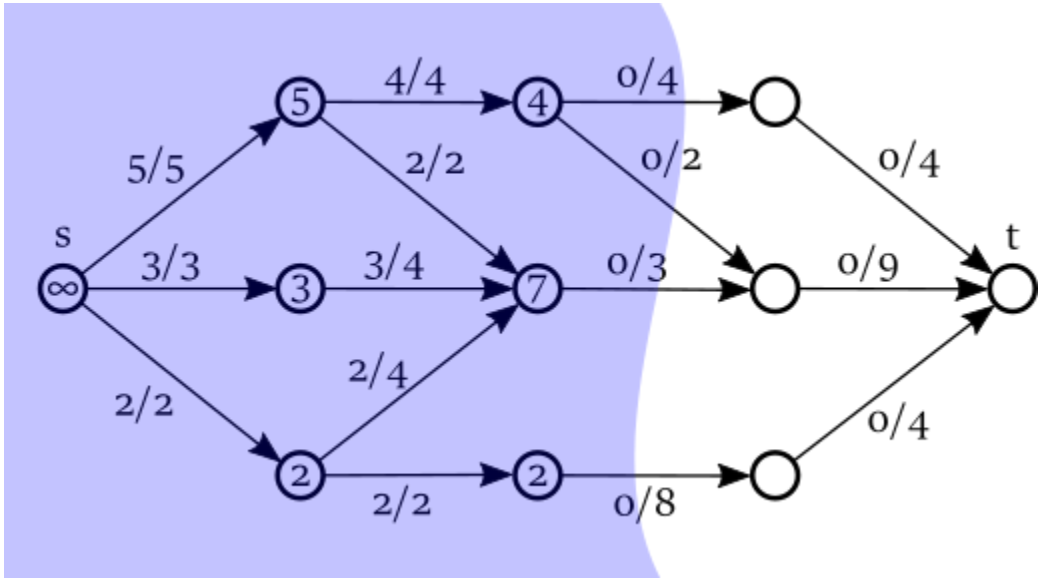
High Tide



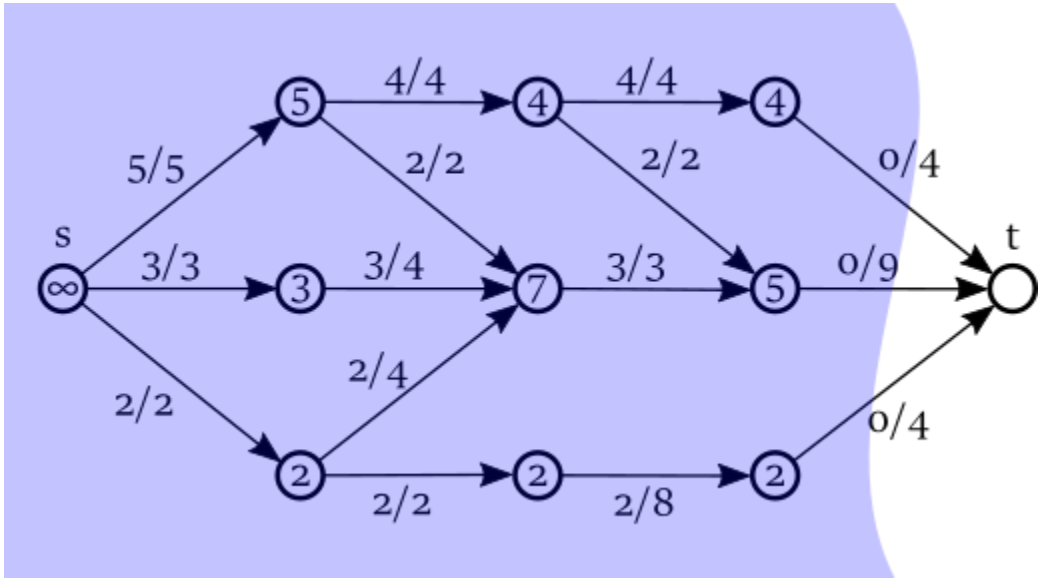
High Tide



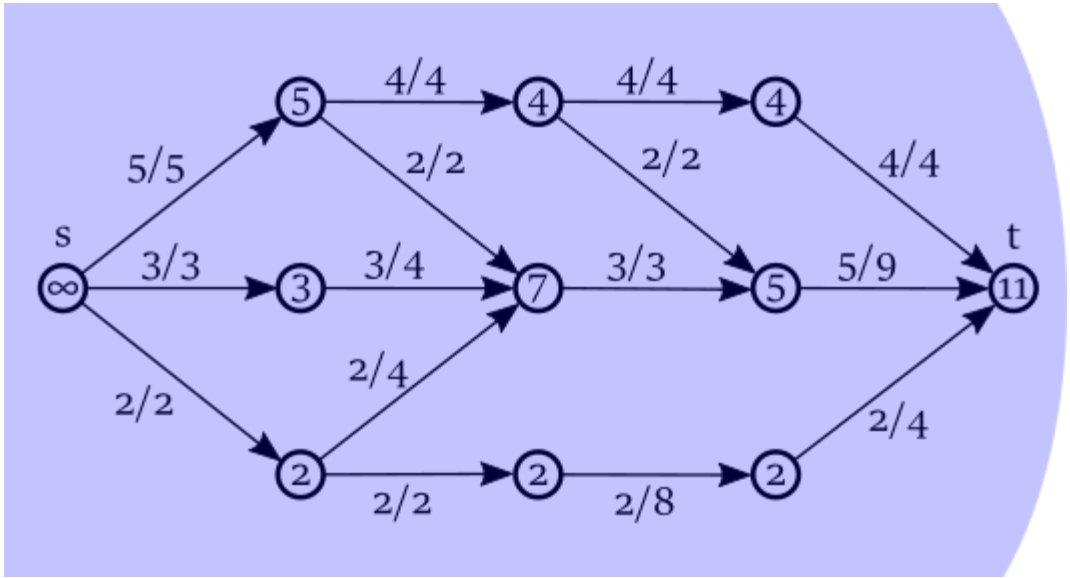
High Tide



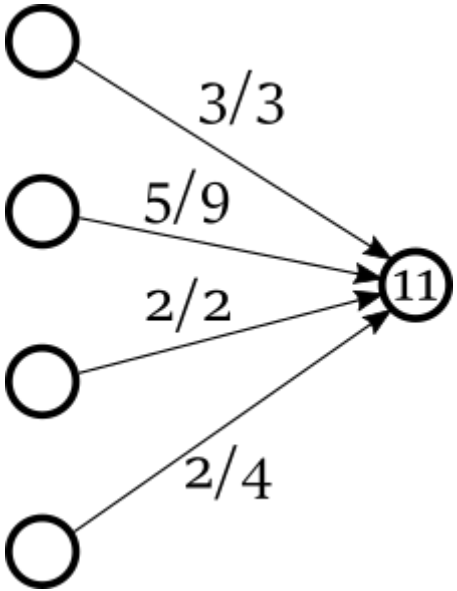
High Tide



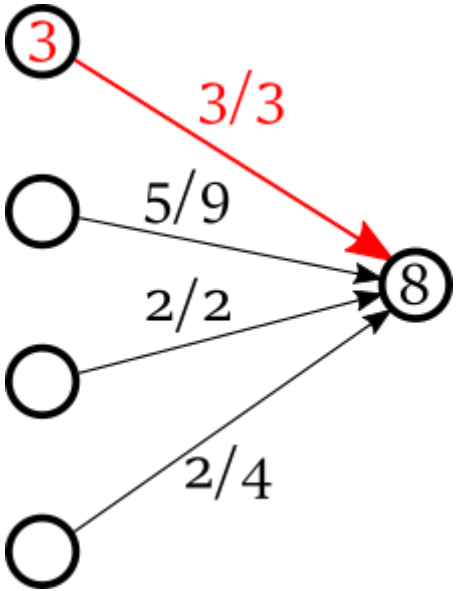
High Tide



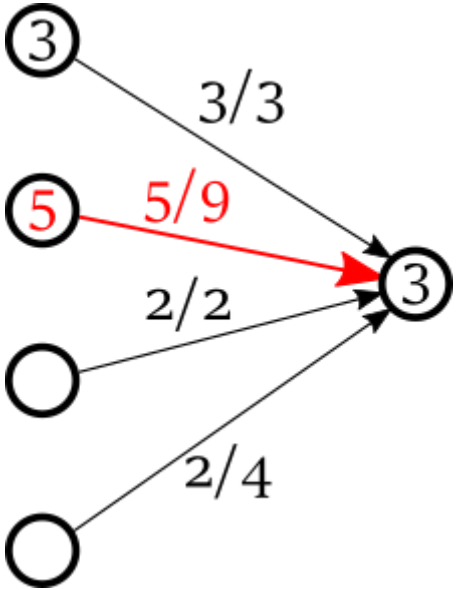
Low Tide



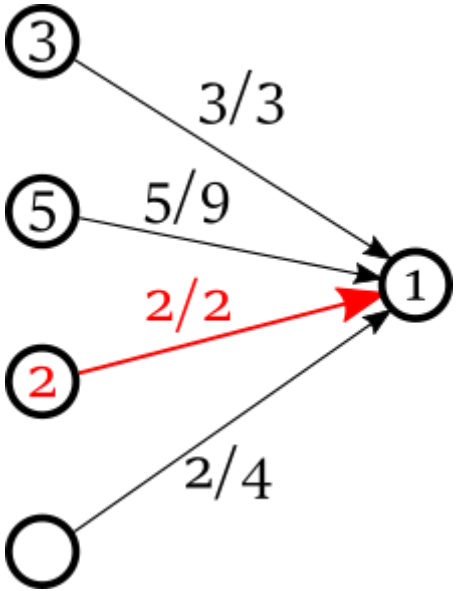
Low Tide



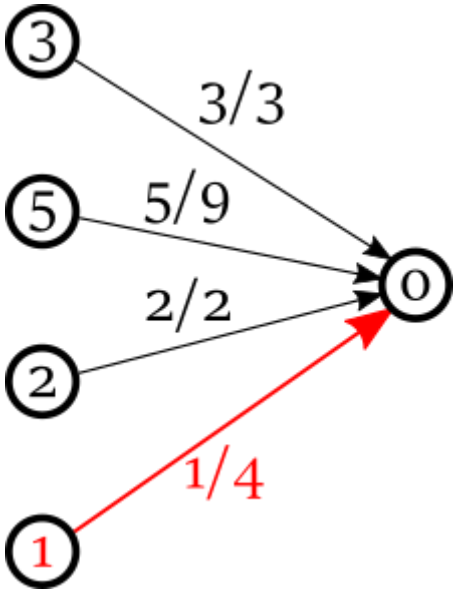
Low Tide



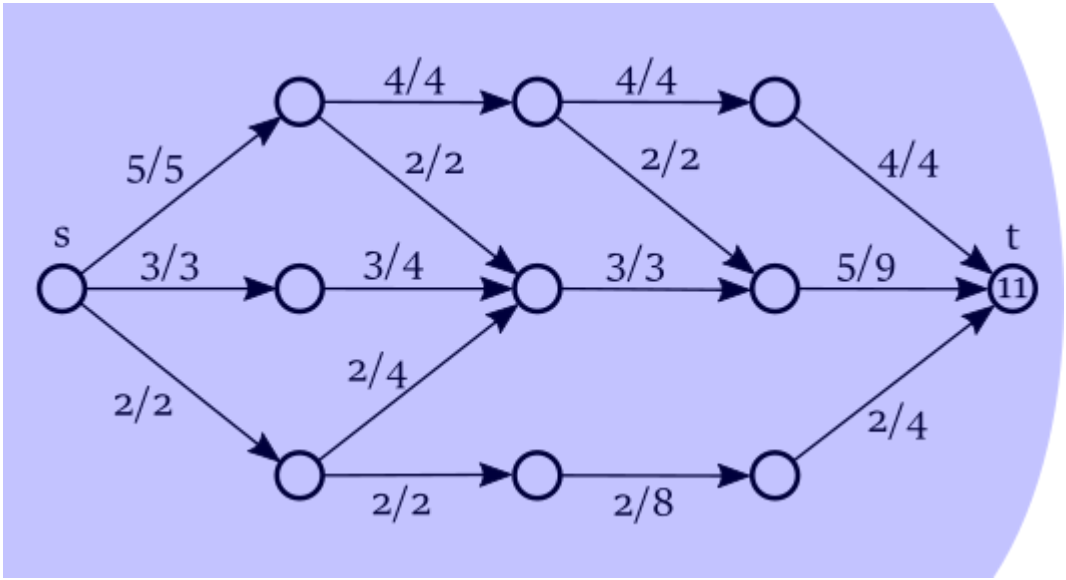
Low Tide



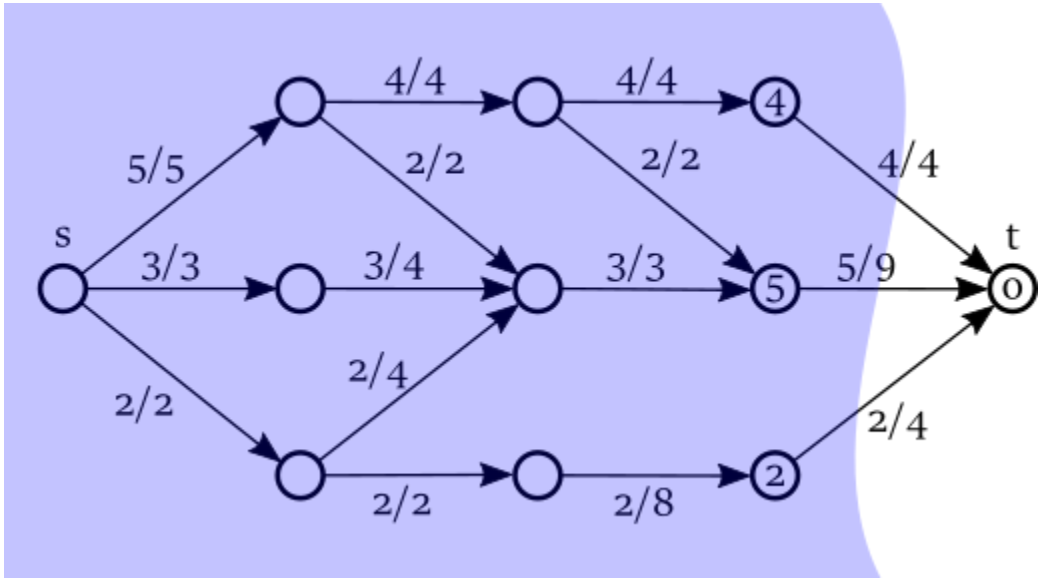
Low Tide



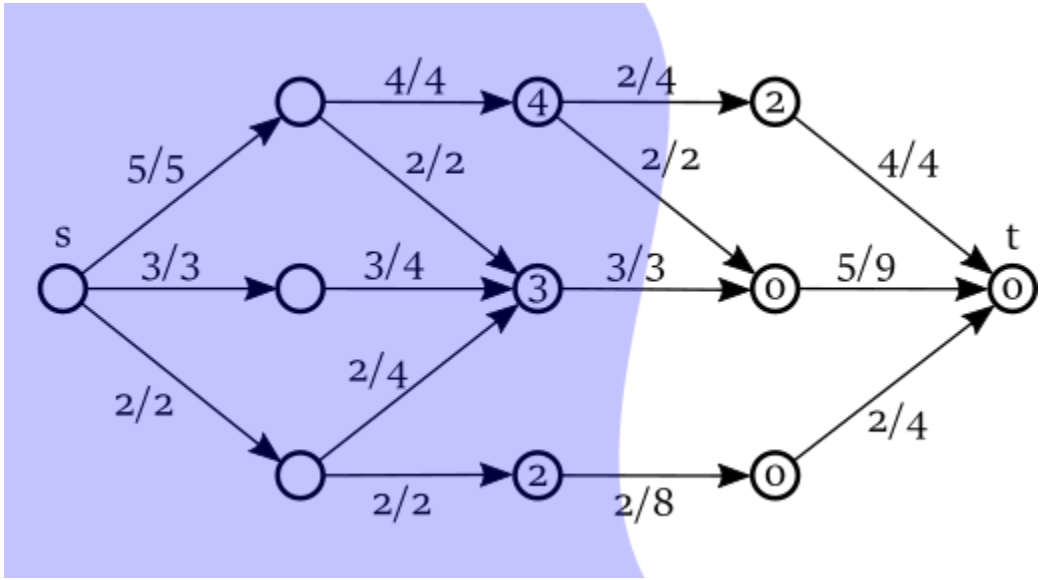
Low Tide



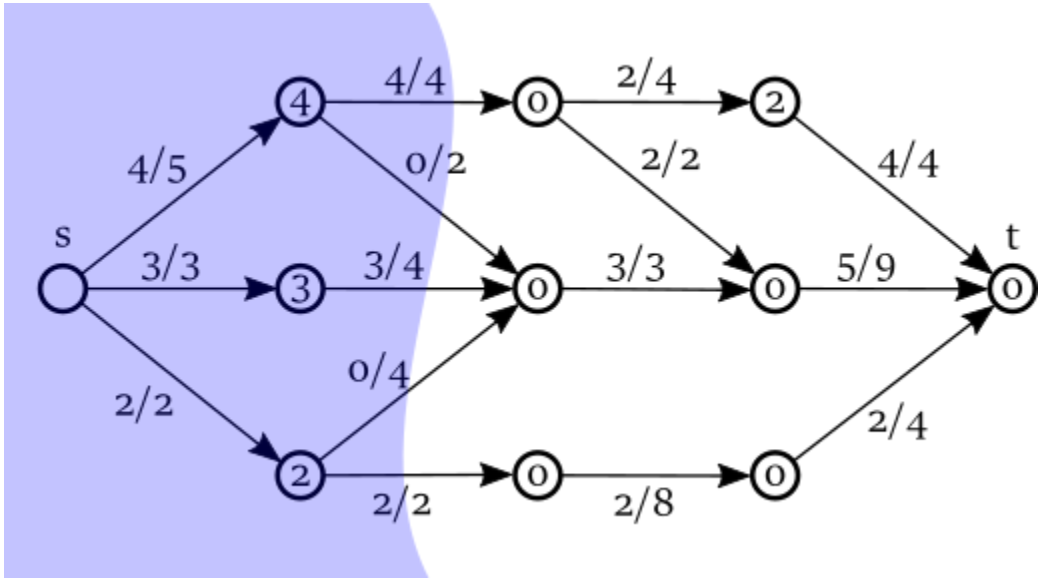
Low Tide



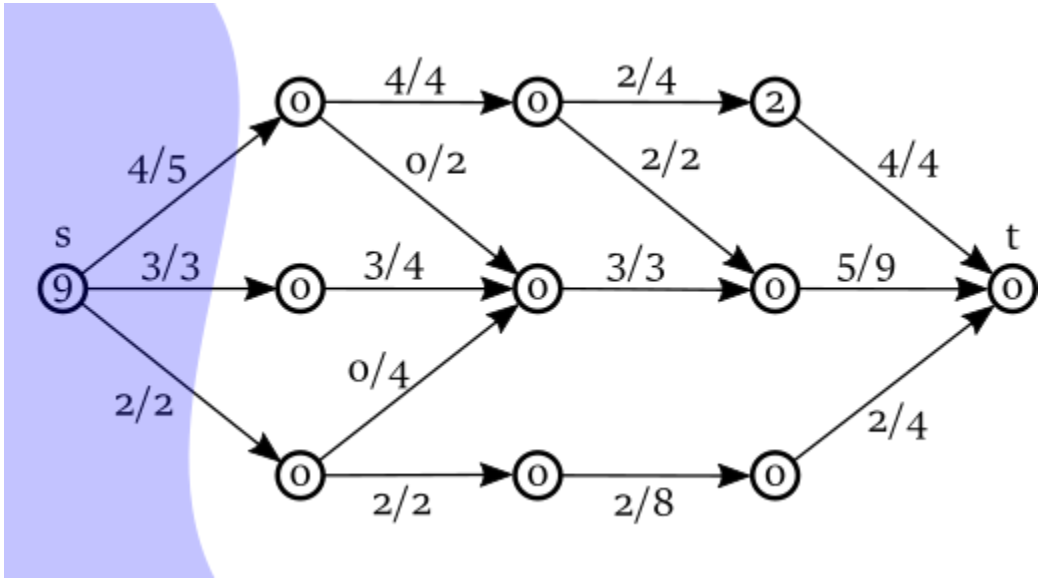
Low Tide



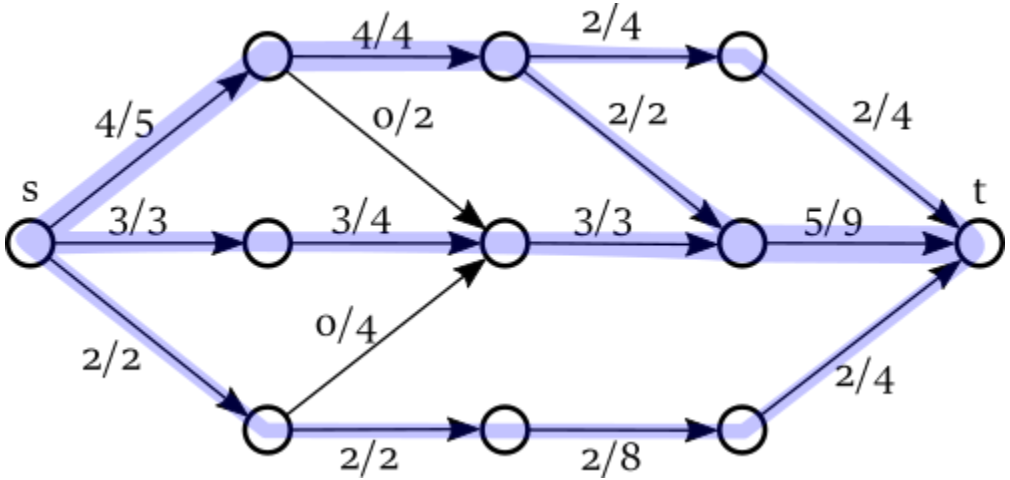
Low Tide



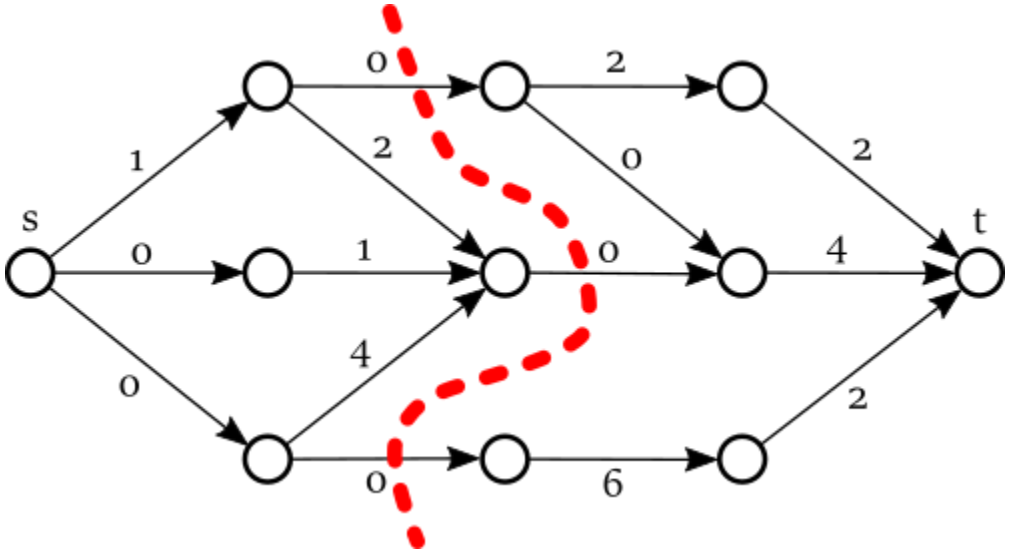
Low Tide



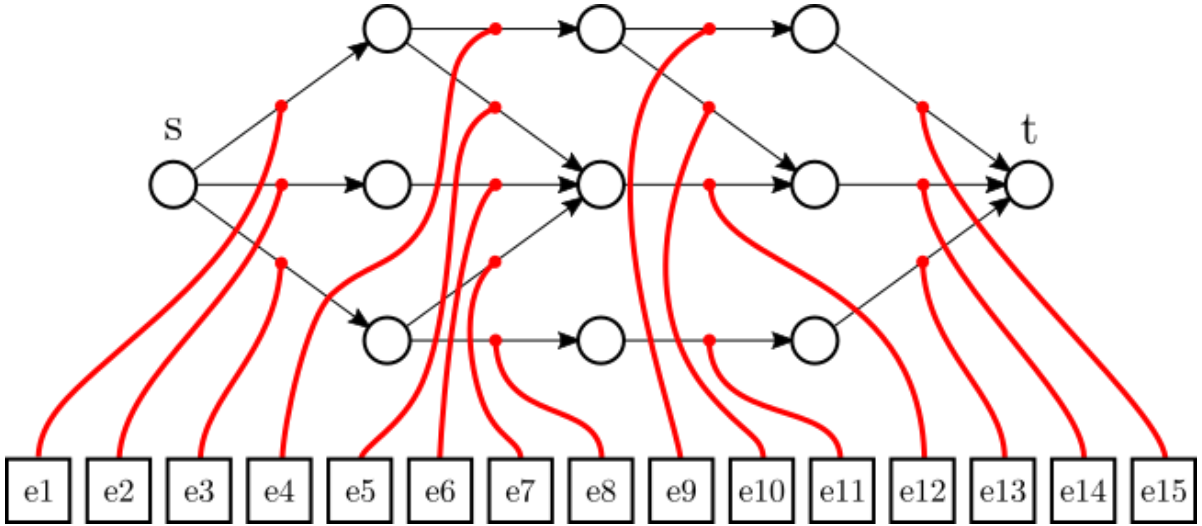
Erosion



Erosion



Implementation



Implementation: High Tide

$h(v) = 0$ for all $v \in V$;
$h(source) \longleftarrow \infty$;
foreach $edge\ e_i(w, v) \in E$ **do**
    $p(e_i) \longleftarrow \min(cap(e_i) - f(e_i), h(w))$;
    $h(v) \longleftarrow h(v) + p(e_i)$;
end

Implementation: Low Tide

$l(v) = 0$ for all $v \in V$
$l(sink) \longleftarrow h(sink)$;
foreach *edge* $e_i(w, v) \in E$ in reverse order **do**
    $p(e_i) \longleftarrow \min(p(e_i), h(w) - l(w), l(v))$;
    $l(v) \longleftarrow l(v) - p(e_i)$;
    $l(w) \longleftarrow l(w) + p(e_i)$;
end

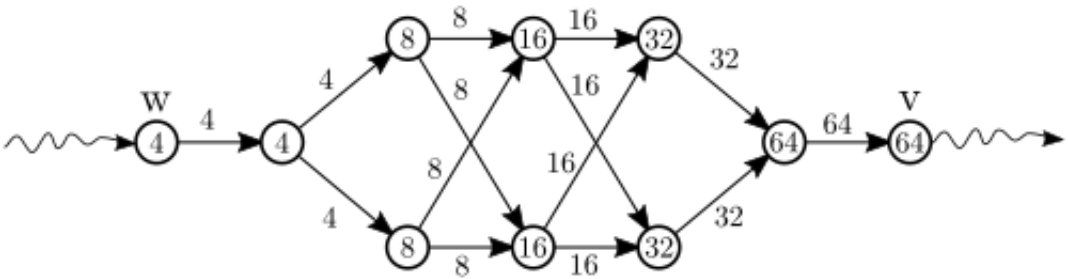
Implementation: Erosion

$h(v) = 0$ for all $v \in V$;
$h(source) \longleftarrow l(source)$;
foreach $edge\ e_i(w, v) \in E$ **do**
    $p(e_i) \longleftarrow \min(p(e_i), h(w))$;
    $h(w) \longleftarrow h(w) - p(e_i)$;
    $h(v) \longleftarrow h(v) + p(e_i)$;
    $f(e_i) \longleftarrow f(e_i) + p(e_i)$;
    $f\big(rev(e_i)\big) \longleftarrow f\big(rev(e_i)\big) - p(e_i)$;
end

Run-time Analysis

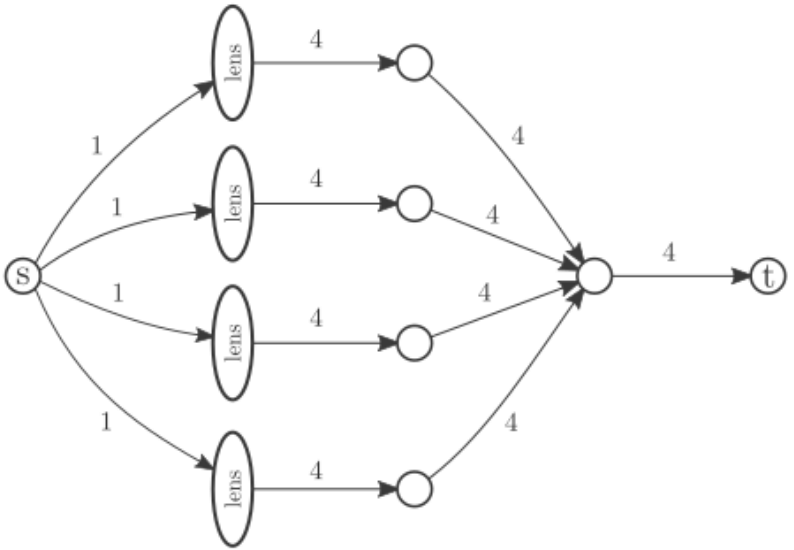
- Shortest Augmenting Path: $O(nm^2)$

Bounding Tide Cycles



Bounding Tide Cycles

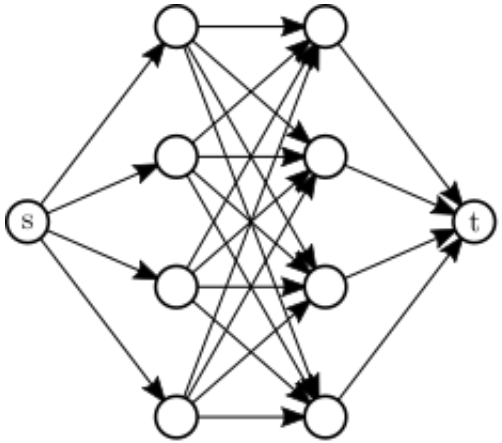
Requires $O\left(\frac{n}{\log(n)}\right)$ tide cycles.

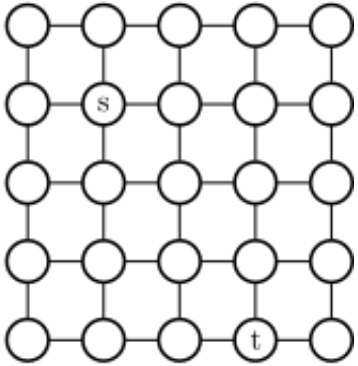


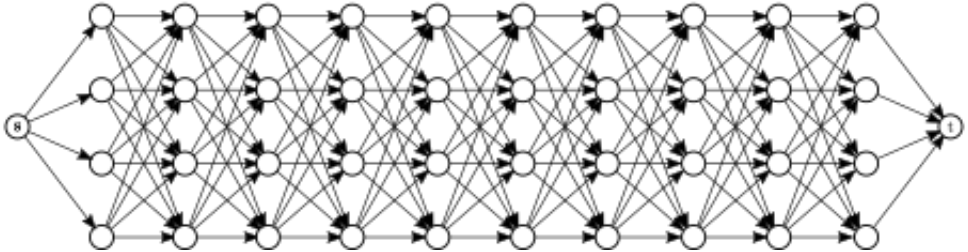
Experiments

- Six suites of tests
- 200 tests per suite (10 different values of n)
- Experiments run against five other algorithms
- Also included a version of Tidal Flow without heuristic function h
- Each algorithm run for a maximum of 20 seconds

Test Suites







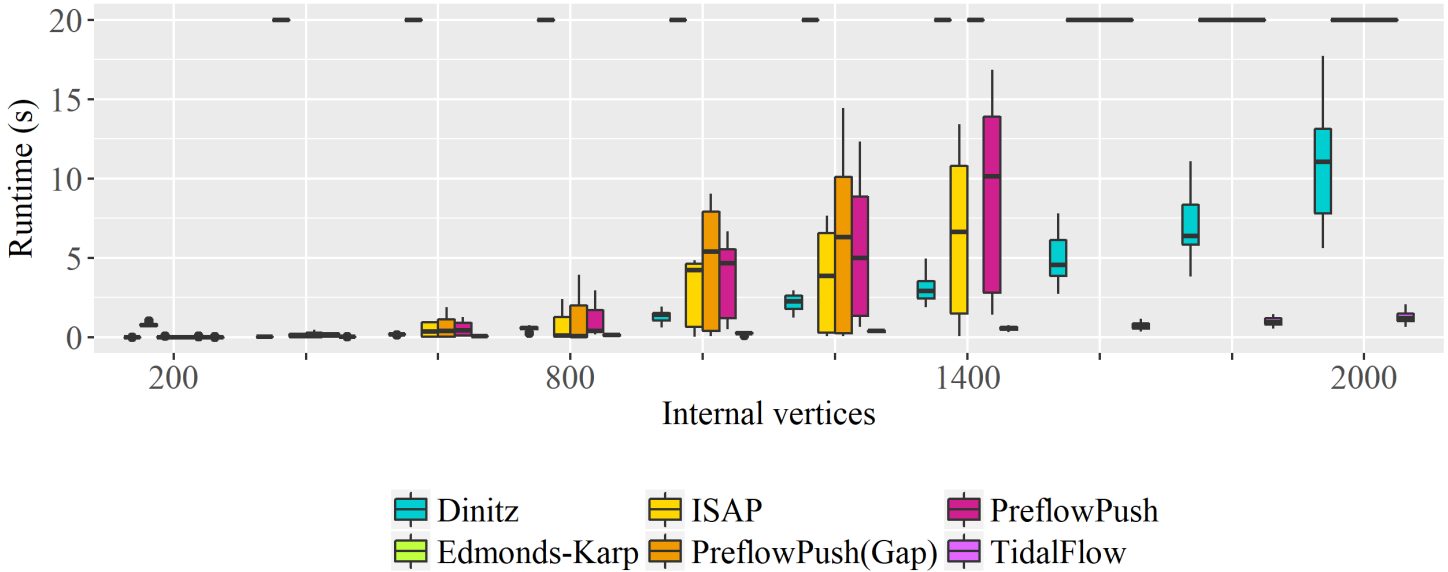
Bipartite Matching Suites

- Sparse versus dense
- Unit versus high capacity

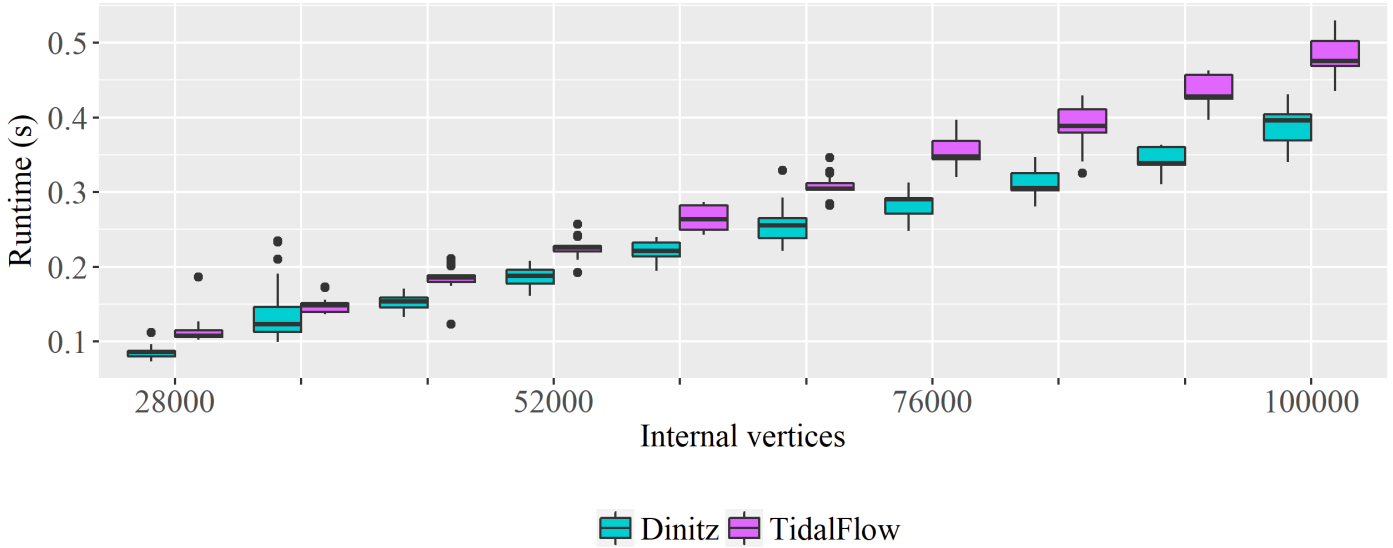
Flow Algorithms

- Shortest Augmenting Path (Edmonds and Karp, 1972)
- Dinitz Algorithm (Dinitz, 1970)
- Preflow-Push (Goldberg and Tarjan, 1988)
- Preflow-Push with Gap Heuristic (Goldberg and Tarjan, 1988)
- Improved Shortest Augmenting Path (Orlin and Ahuja, 1987)

Results: Dense High Capacity BPM



Results: Sparse Unit BPM



Open Questions

- Empirical study is only preliminary
- Bounding tide cycles: $O\left(\frac{n}{\log(n)}\right)$ versus $O(m)$
- Empirical results suggest $O(nm^2)$ runtime is not tight
- Does a better h exist?

Questions?