EFFICIENT MAXIMUM FLOW ALGORITHM

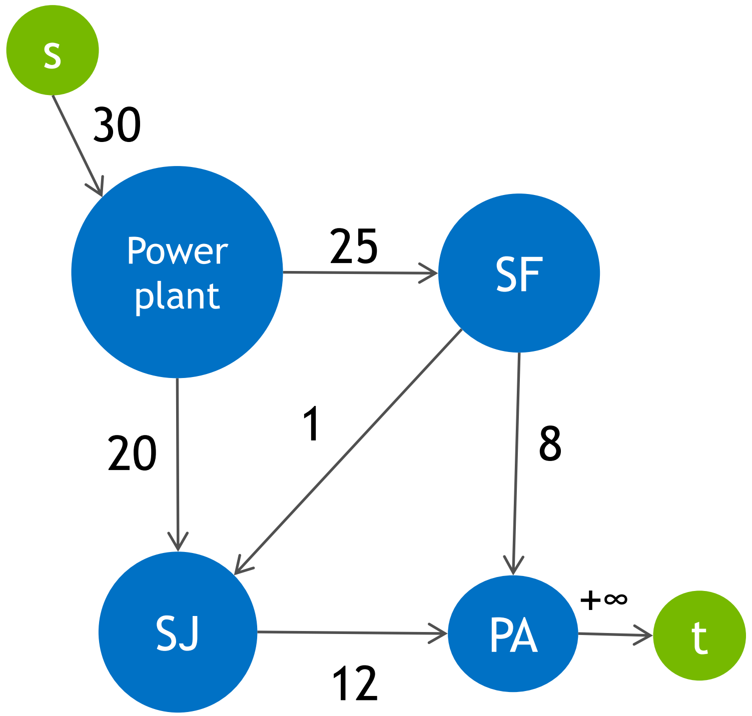
Nikolay Sakharnykh – Hugo Braun; May 10, 2017

NVIDIA

MAXIMUM FLOW

Definition

Example: How much instant power can Palo Alto get using that electric grid?



- Directed graph
- Flow capacities on edges
- Maximum flow from s to t?

MAXIMUM FLOW

Applications

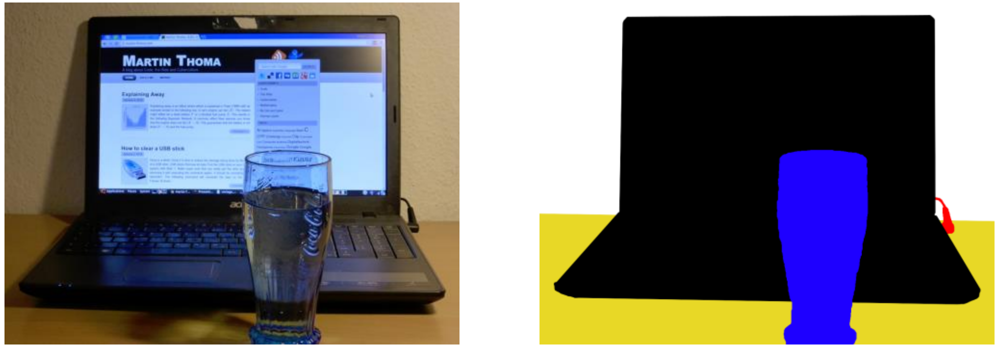
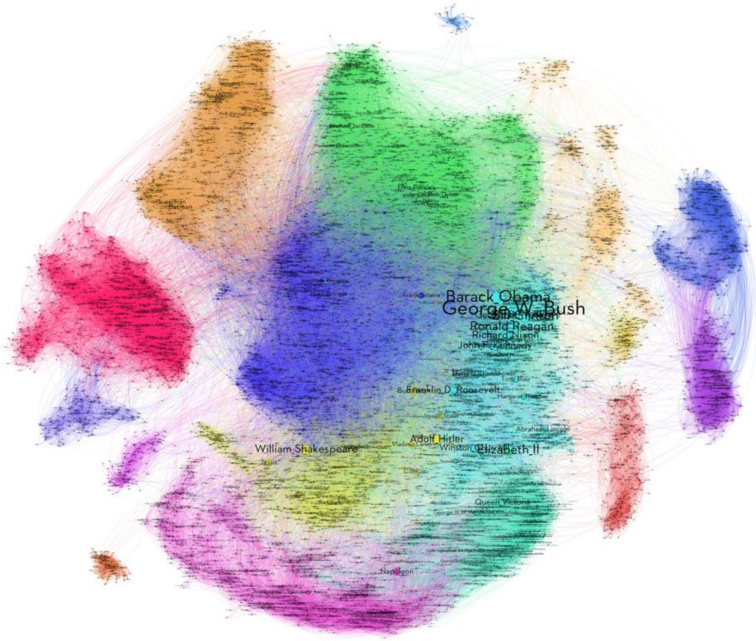


Image segmentation



Community detection

MAXIMUM FLOW SOLVERS

AUGMENTING PATHS

Iteratively find a new augmenting path

- Ford–Fulkerson
- Edmonds-Karp
- Dinic's/MPM

PREFLOW

Push flow locally in a preflow graph

- Push relabel and its variants

OTHER

Linear programming

AGENDA

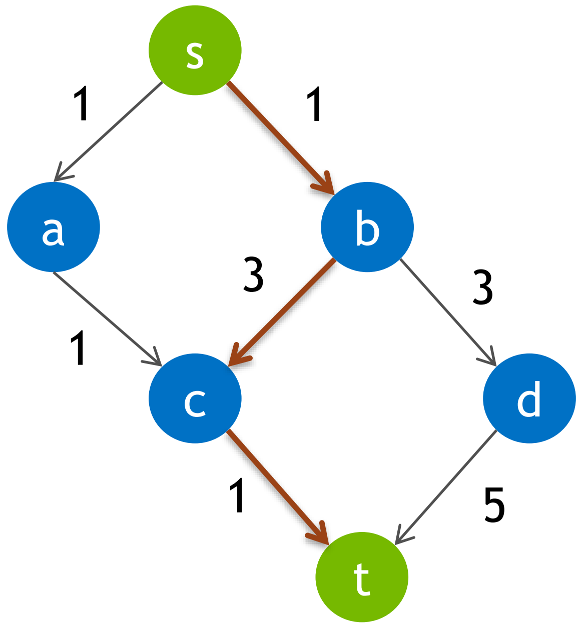
Edmonds-Karp

Push-relabel

MPM

FORD FULKERSON

Workflow

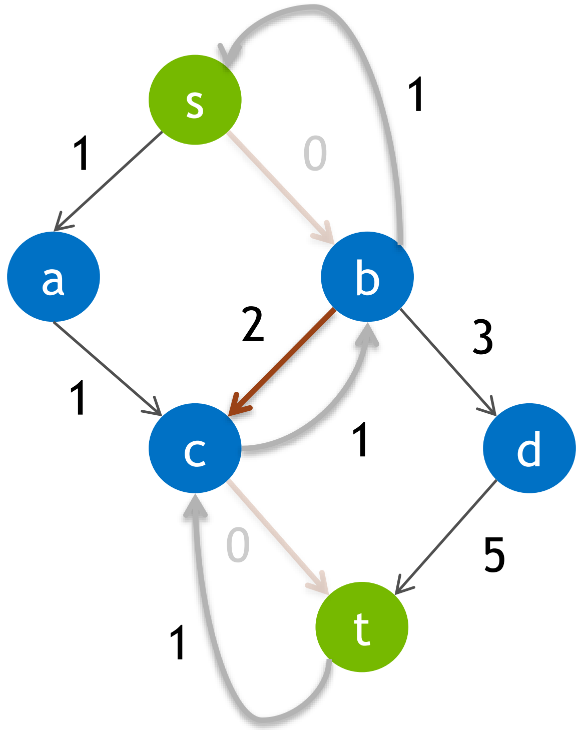



```
while a path p of capacity c > 0 exists from s to t:
    maxflow += c
    for all edges in path p:
        edge.capacity -= c
        add reverse edge from edge.destination to
            edge.source of capacity c
```

→ Augmenting path on first iteration, path of capacity min(1,3,1) = 1

FORD FULKERSON

Workflow

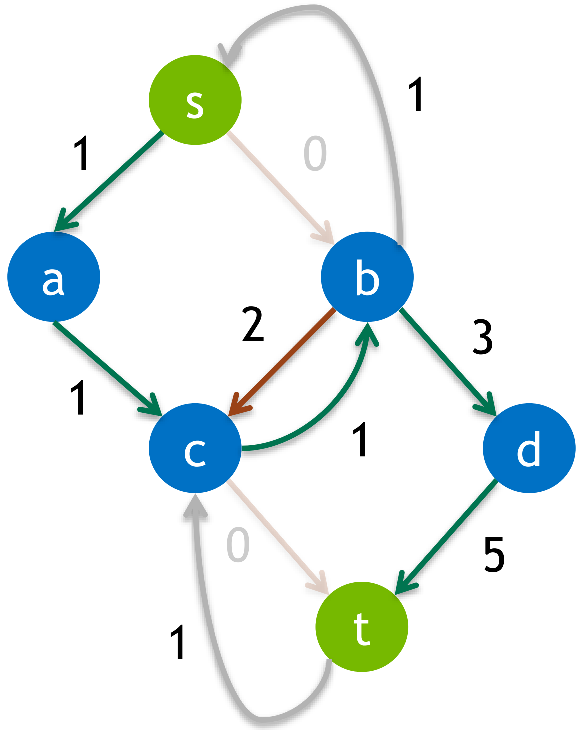



```
while a path p of capacity c > 0 exists from s to t:
    maxflow += c
    for all edges in path p:
        edge.capacity -= c
        add reverse edge from edge.destination to
           edge.source of capacity c
```

 Augmenting path on first iteration, path of capacity min(1,3,1) = 1

FORD FULKERSON

Workflow

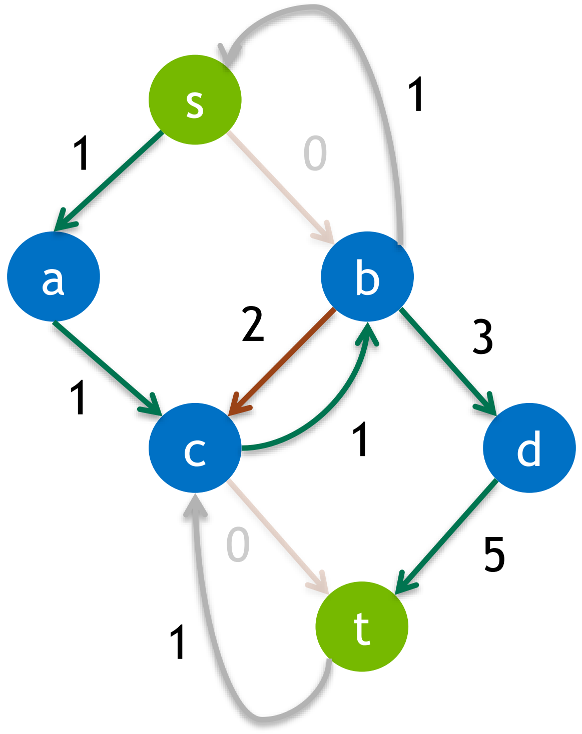



```
while a path p of capacity c > 0 exists from s to t:
    maxflow += c
    for all edges in path p:
        edge.capacity -= c
        add reverse edge from edge.destination to
           edge.source of capacity c
```

 Augmenting path on first iteration, path of capacity min(1,3,1) = 1

 Augmenting path on second iteration, path of capacity min(1,1,1,3,5) = 1

EDMONDS-KARP



Edmonds-Karp: variation of Ford Fulkerson

Main idea: use *the shortest* augmenting path

One augmenting path needs one BFS

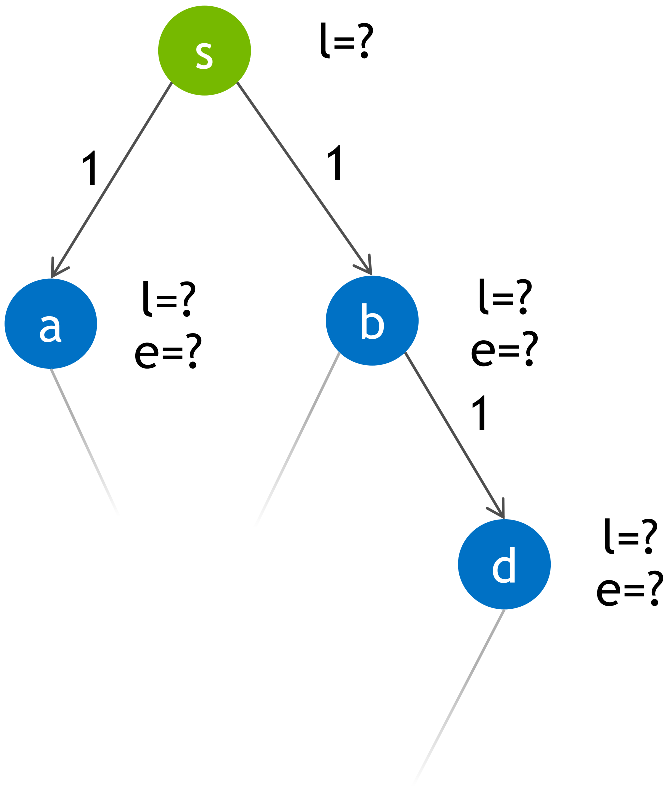
Wikipedia graph: ~5000 augmenting paths

→ Ford-Fulkerson & variants use too many graph traversals

PUSH-RELABEL

PUSH-RELABEL

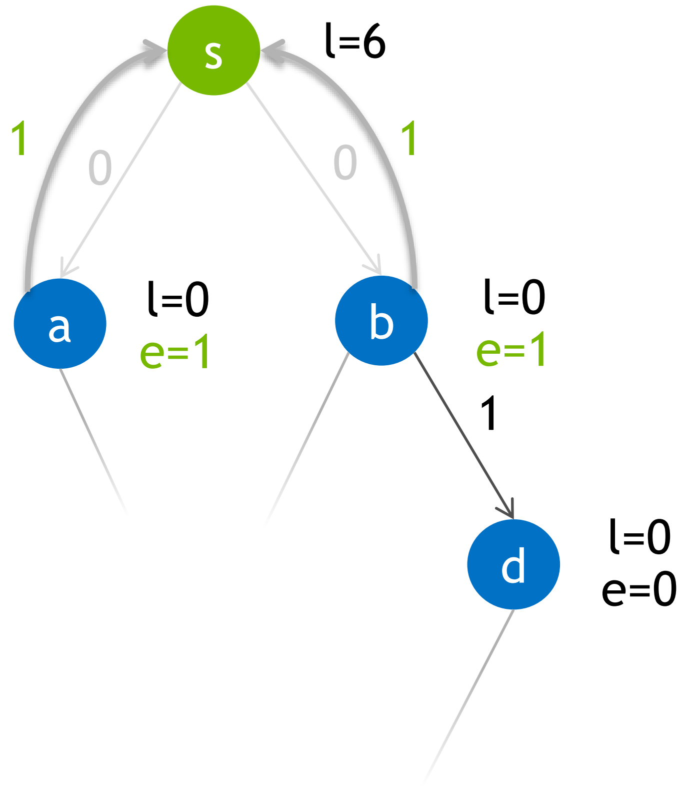
Workflow




```
Saturate all out-edges of s, create reverse edges
    ℓ[s] = number of vertices
    ℓ[v] = 0 for all v ∈ V \ {s}
    while there is an applicable push or relabel operation
        execute the operation

push(u, v):
     if(e[u] > 0 and ℓ[u] == ℓ[v] + 1)
         push e[u] amount of flow from u to v

relabel(u):
    if(e[u] > 0 and ℓ[u] <= ℓ[v] for all current neighbors)
         ℓ[u] = minimum ℓ[v] among neighbors + 1
```

PUSH-RELABEL

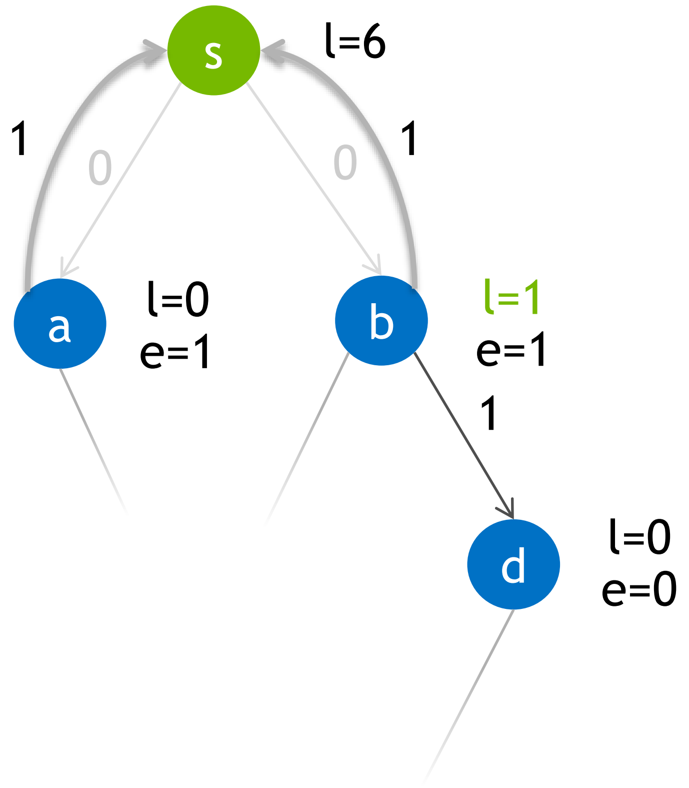
Workflow




```
Saturate all out-edges of s, create reverse edges
    ℓ[s] = number of vertices
    ℓ[v] = 0 for all v ∈ V \ {s}
    while there is an applicable push or relabel operation
        execute the operation

push(u, v):
     if(e[u] > 0 and ℓ[u] == ℓ[v] + 1)
         push e[u] amount of flow from u to v

relabel(u):
    if(e[u] > 0 and ℓ[u] <= ℓ[v] for all current neighbors)
         ℓ[u] = minimum ℓ[v] among neighbors + 1
```

PUSH-RELABEL

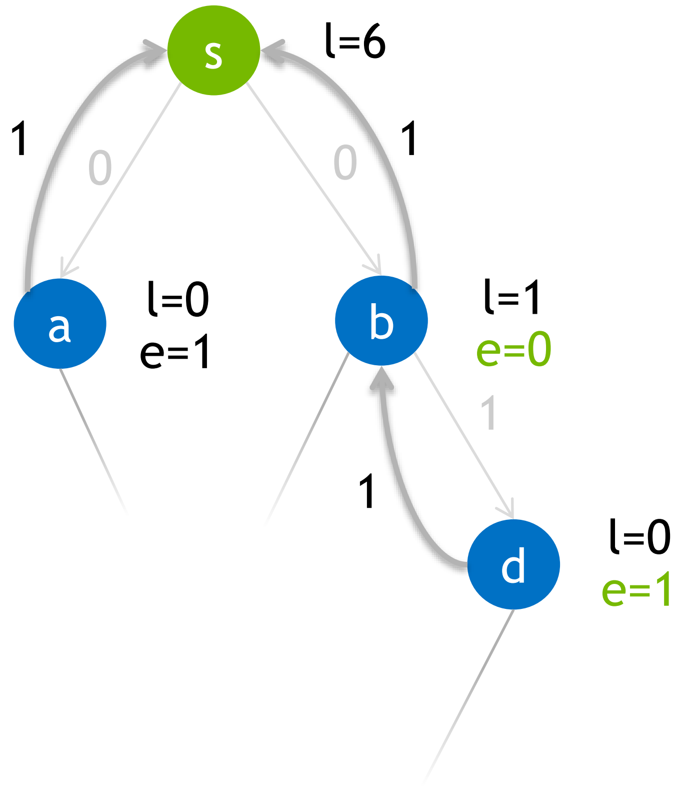
Workflow




```
Saturate all out-edges of s, create reverse edges
    ℓ[s] = number of vertices
    ℓ[v] = 0 for all v ∈ V \ {s}
    while there is an applicable push or relabel operation
        execute the operation

push(u, v):
     if(e[u] > 0 and ℓ[u] == ℓ[v] + 1)
         push e[u] amount of flow from u to v

relabel(u):
    if(e[u] > 0 and ℓ[u] <= ℓ[v] for all current neighbors)
         ℓ[u] = minimum ℓ[v] among neighbors + 1
```

PUSH-RELABEL

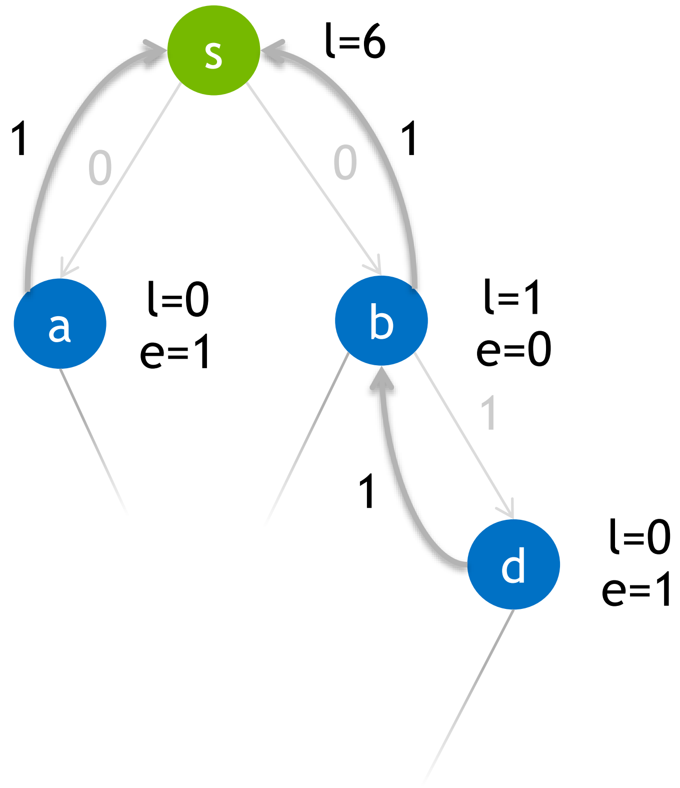
Workflow




```
Saturate all out-edges of s, create reverse edges
    ℓ[s] = number of vertices
    ℓ[v] = 0 for all v ∈ V \ {s}
    while there is an applicable push or relabel operation
        execute the operation

push(u, v):
     if(e[u] > 0 and ℓ[u] == ℓ[v] + 1)
         push e[u] amount of flow from u to v

relabel(u):
    if(e[u] > 0 and ℓ[u] <= ℓ[v] for all current neighbors)
         ℓ[u] = minimum ℓ[v] among neighbors + 1
```

PUSH-RELABEL

Parallelism issues



Source of parallelism:

```
while there is an applicable push or relabel operation
        execute the operation
```

At this step, we could relabel **a** or **d**. Which one?

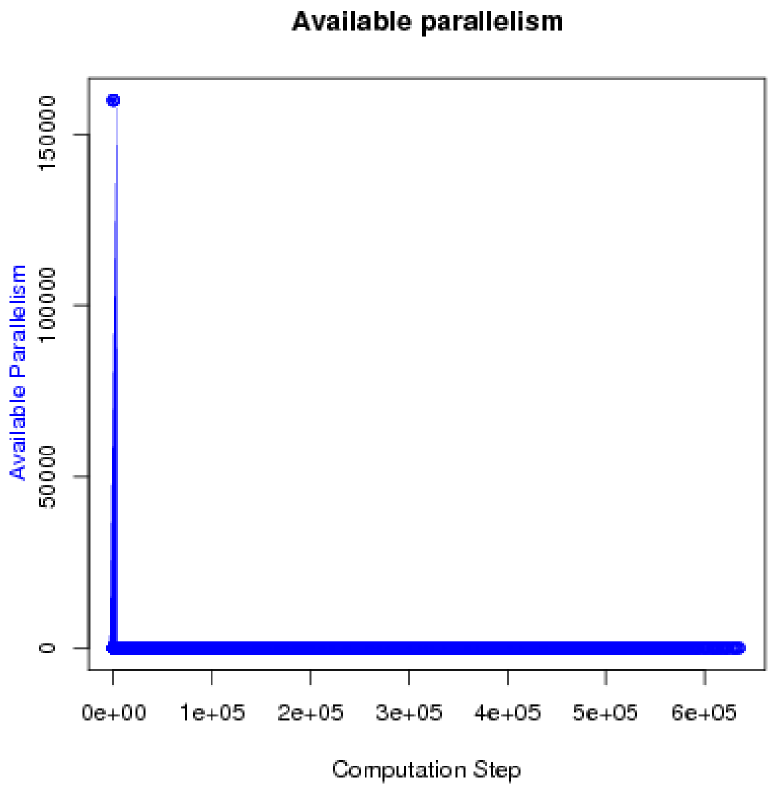
Complexity of heuristics :

PRIORITY	LARGEST L	SMALLEST L	FIFO
Complexity	$O(V^2\sqrt{E})$	$O(V^2E)$	$O(V^3)$

Order affects convergence. Massive parallelism yields random order

PUSH-RELABEL

Parallelism issues



Source : The University of Texas at Austin

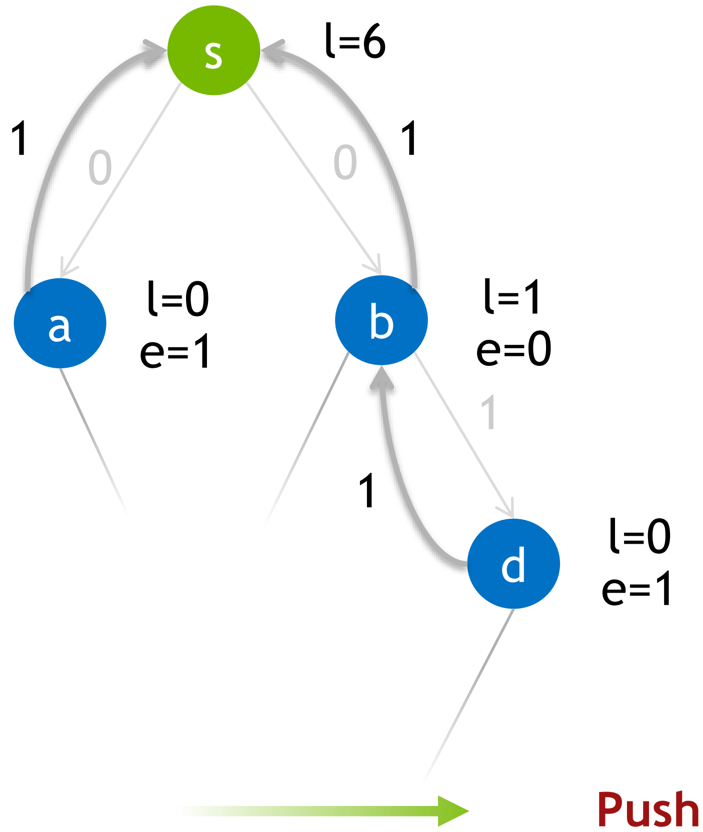
In theory,
number of threads = number of vertices

In practice,
number of active vertices << number of vertices

Parallelism drops. Not enough to saturate the GPU

PUSH-RELABEL

Conclusion



- Actual parallelism is **low**
- Massive parallelism yields **random order** which damages performance
- We need graph traversals (BFS) for some critical heuristics

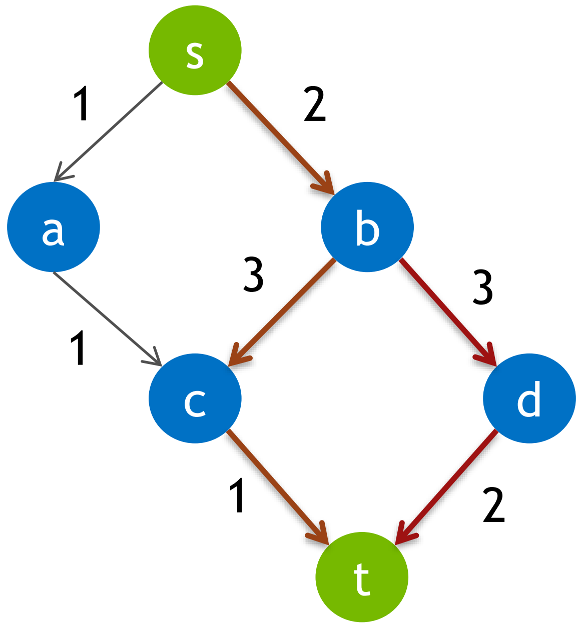
road_usa: GPU does 20 BFS, CPU does only 3 BFS
CPU is faster since it requires fewer traversals

Push-relabel not suited for GPU implementation

MPM

DINIC'S

Workflow



Two augmenting paths of length 3

They have been discovered using just one BFS

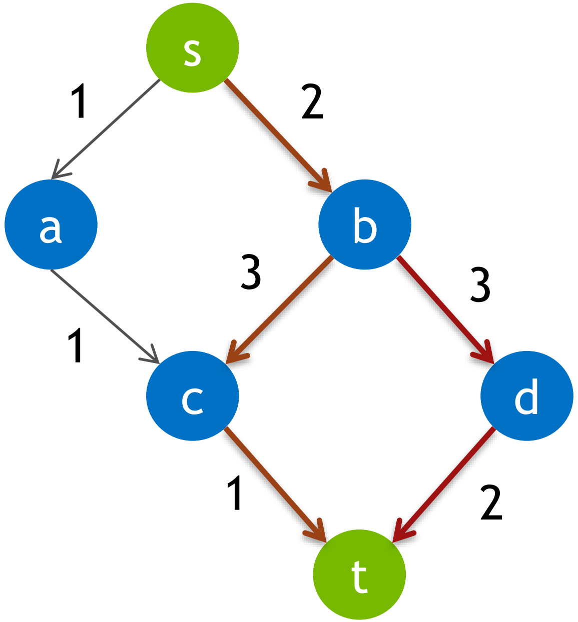
Avoid running BFS twice here

Main idea of Dinic's: *reuse* BFS results

Edges on paths of length 3

DINIC'S

Workflow

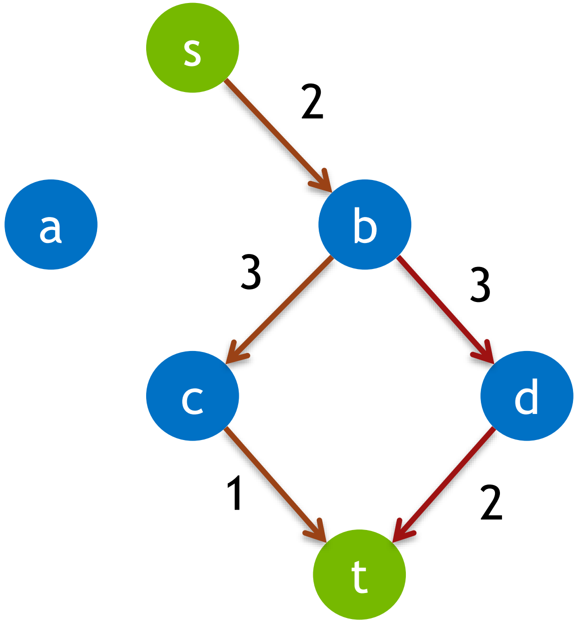


Graph G

```
While s is still connected to t in G, do
     Create layer graph G_L containing only
     shortest paths from s to t

     While s is still connected to t, do
          Find augmenting path from s to t in G_L
          Push corresponding flow in G_L, update edges
```

DINIC'S

Workflow

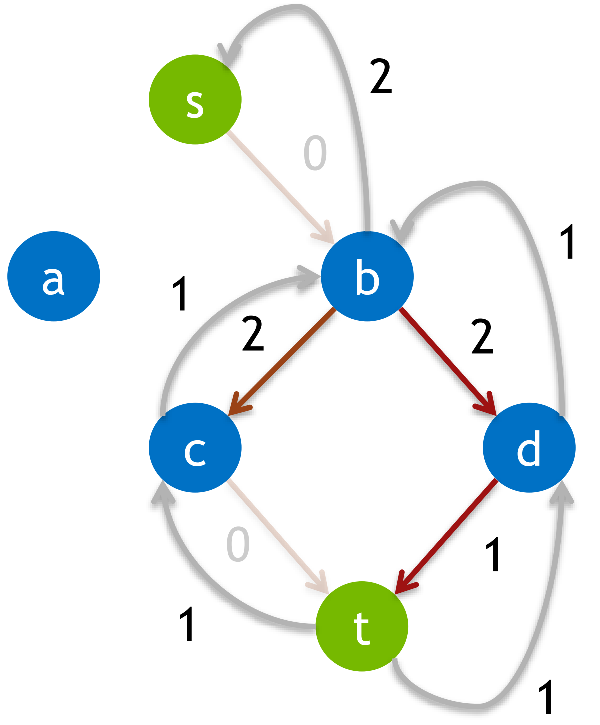


Graph $G_{L(3)}$

```
While s is still connected to t in G, do
     Create layer graph G_L containing only
     shortest paths from s to t

     While s is still connected to t, do
          Find augmenting path from s to t in G_L
          Push corresponding flow in G_L, update edges
```

DINIC'S

Workflow



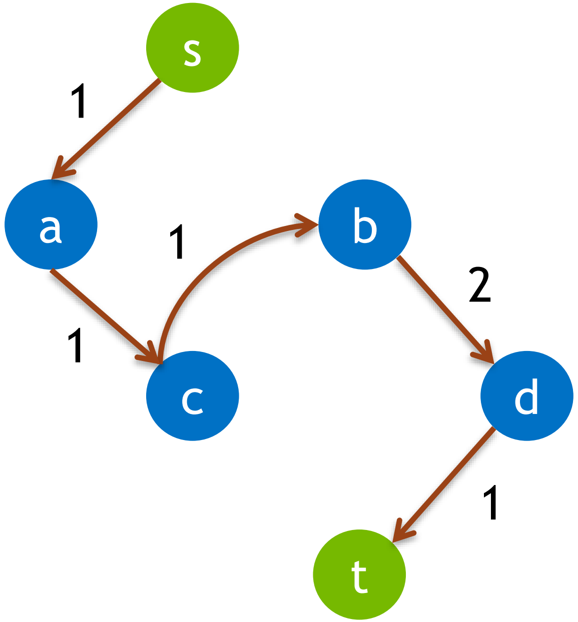
Graph $G_{L(3)}$

```
While s is still connected to t in G, do
     Create layer graph G_L containing only
     shortest paths from s to t

     While s is still connected to t, do
          Find augmenting path from s to t in G_L
          Push corresponding flow in G_L, update edges
```

DFS

DINIC'S

Workflow

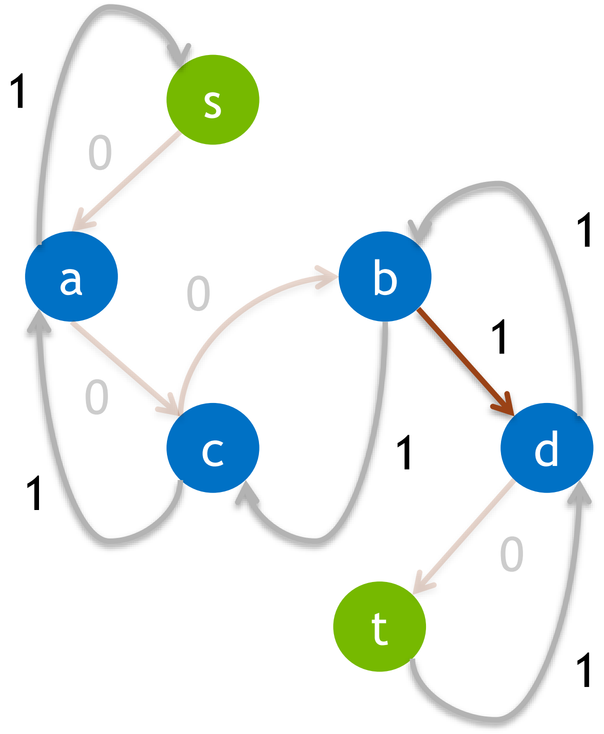


Graph $G_{L(4)}$

```
While s is still connected to t in G, do
     Create layer graph G_L containing only
     shortest paths from s to t

     While s is still connected to t, do
          Find augmenting path from s to t in G_L
          Push corresponding flow in G_L, update edges
```

DINIC'S

Workflow



Graph $G_{L(4)}$

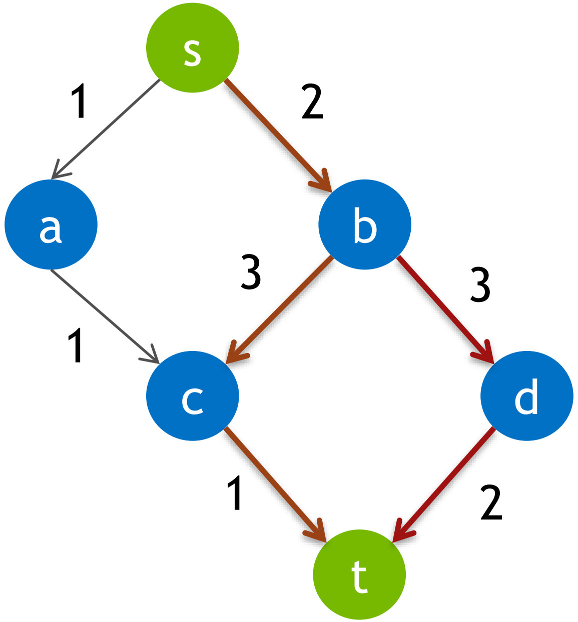
```
While s is still connected to t in G, do
     Create layer graph G_L containing only
     shortest paths from s to t

     While s is still connected to t, do
          Find augmenting path from s to t in G_L
          Push corresponding flow in G_L, update edges
```

DFS

DFS traverse all vertices on GPU
We lose all advantages of Dinic's

MPM

Workflow

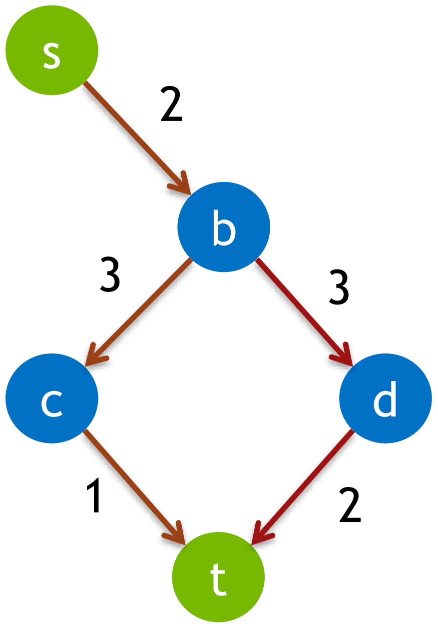


Graph G

```
While s is still connected to t in G, do
     Create layer graph G_L containing only
     shortest paths from s to t

     While s is still connected to t, do
          Find vertex u with minimum potential m, with
          potential(u) = min(degree_in(u), degree_out(u))

          push m from u to t, pull m from s to u
          remove all vertex with min(degree_in(u), degree_out(u))=0
```

MPM

Workflow

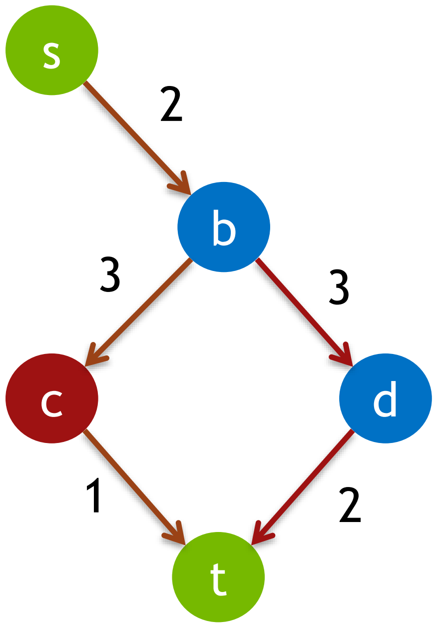


Graph G

```
While s is still connected to t in G, do
     Create layer graph G_L containing only
     shortest paths from s to t

     While s is still connected to t, do
          Find vertex u with minimum potential m, with
          potential(u) = min(degree_in(u), degree_out(u))

          push m from u to t, pull m from s to u
          remove all vertex with min(degree_in(u), degree_out(u))=0
```

MPM

Workflow



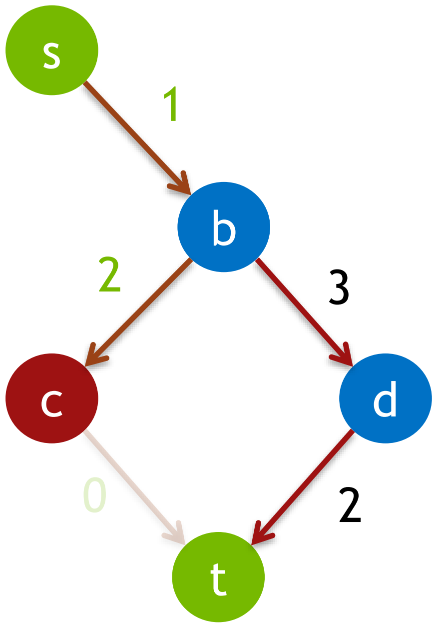
Graph $G_{L(3)}$

```
While s is still connected to t in G, do
    Create layer graph G_L containing only
    shortest paths from s to t

    While s is still connected to t, do
        Find vertex u with minimum potential m, with
        potential(u) = min(degree_in(u), degree_out(u))

        push m from u to t, pull m from s to u
        remove all vertex with min(degree_in(u), degree_out(u))=0
```

is selected so that we know 1 amount of flow will pass through

MPM

Workflow



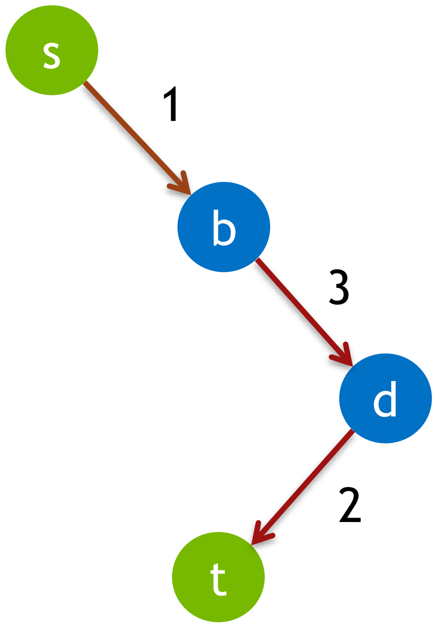
Graph $G_{L(3)}$

```
While s is still connected to t in G, do
    Create layer graph G_L containing only
    shortest paths from s to t

    While s is still connected to t, do
        Find vertex u with minimum potential m, with
        potential(u) = min(degree_in(u), degree_out(u))

        push m from u to t, pull m from s to u
        remove all vertex with min(degree_in(u), degree_out(u))=0
```

c is selected so that we know 1 amount of flow will pass through

MPM

Workflow

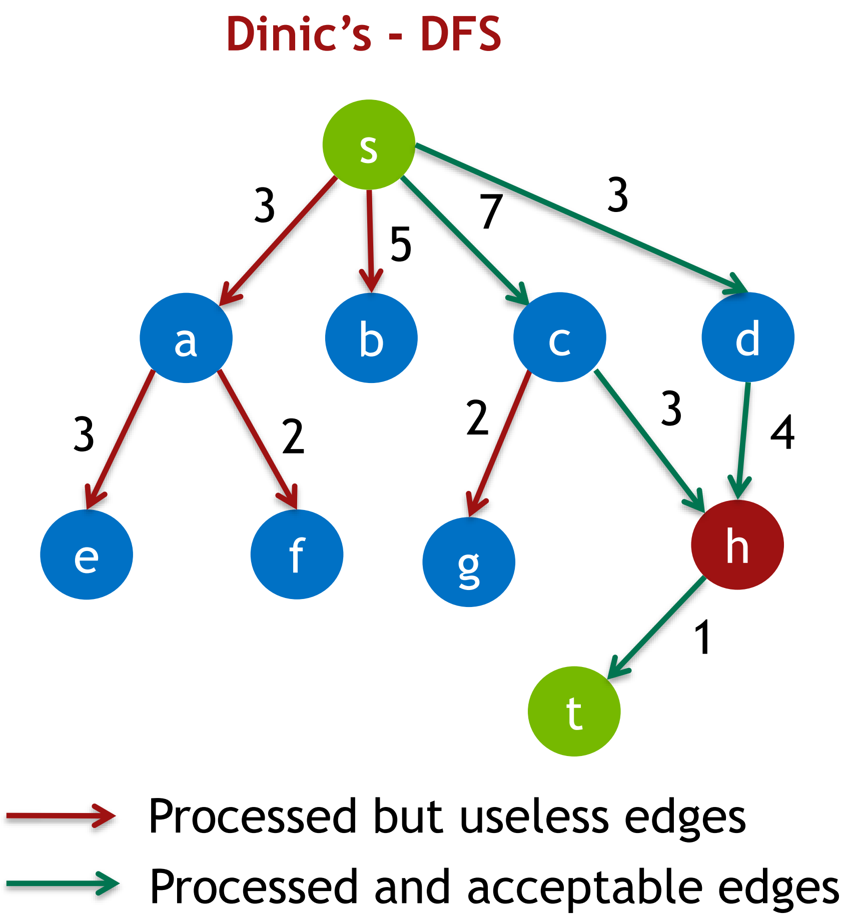


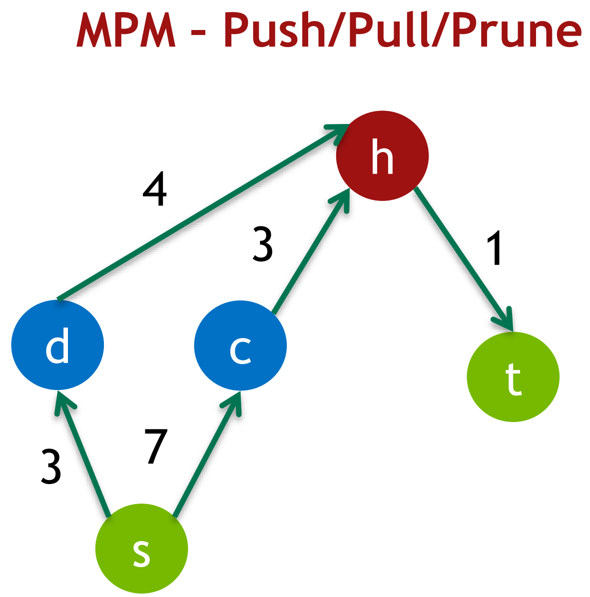
Graph $G_{L(3)}$

```
While s is still connected to t in G, do
    Create layer graph G_L containing only
    shortest paths from s to t

    While s is still connected to t, do
        Find vertex u with minimum potential m, with
        potential(u) = min(degree_in(u), degree_out(u))

        push m from u to t, pull m from s to u
        remove all vertex with min(degree_in(u), degree_out(u))=0
```

MPM

Dinic's vs MPM

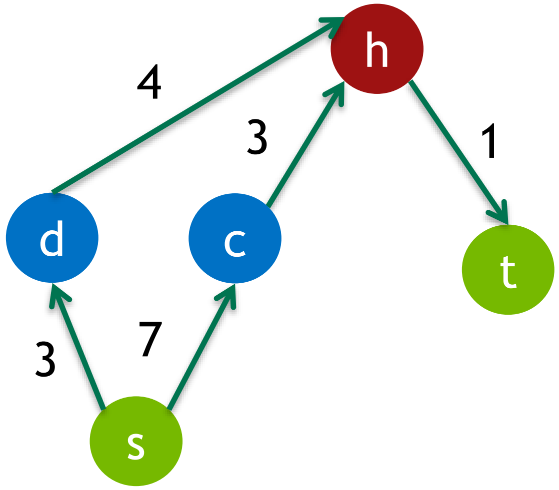




Across the graph, min potential = 1 (vertex h)
Pushing 1 to t, pulling 1 from s, using any edges

MPM

Dinic's vs MPM



Saturating one augmenting path on GPU:

- **MPM**: Push/pull/prune process 30us
- **Edmonds-Karp/Dinic's**: one BFS >1ms

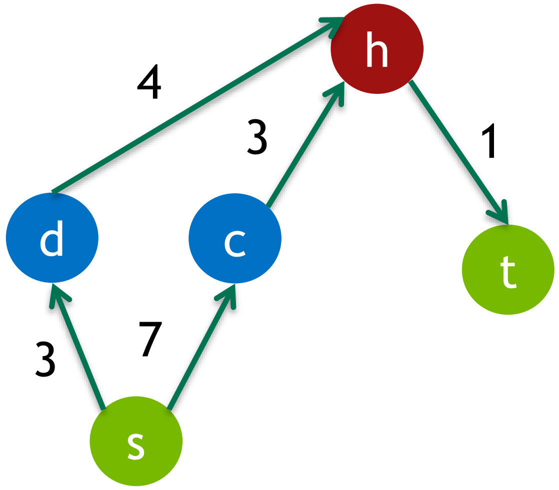
Perf. bounded by kernel launch latency

Example: Wikipedia 2011

- MPM: 5 BFS, 6000 augmenting paths
- EK: 6000 BFS

MPM

GPU design



MPM paper gives a high level implementation

Most of the work went into GPU implementation design (2 out of 3 months)

MAXIMUM FLOW RESULTS

Galois on dual socket Haswell 16 cores vs NVIDIA Titan X (Pascal)

GRAPH	N	NNZ	SPEED UP		
			AVG	MIN	MAX
wiki03	455436	3811198	9.1	1.7	15.3
wiki11	3721339	121043107	22.5	19.8	28.9
road_usa	23947347	57708624	2	0.7	4.2
road CA	1971281	5533214	2.3	0.8	4.9

EFFICIENT MAXIMUM FLOW ALGORITHM

Takeaways

- **Black-box solver**: large variety of applications can be seen as the flow problem.

- **Data-dependent, irregular algorithm:** how to create enough "real" parallelism and how to avoid latency issues on the GPU.

- **Order of magnitude speed-ups** on *wide graphs*. Long graphs require a more efficient graph traversal implementation.

REFERENCES

- An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision, Yuri Boykov, Vladimir Kolmogorov
- Finding Web Communities by Maximum Flow Algorithm using Well-Assigned Edge Capacities, Noriko IMAFUJI, and Masaru KITSUREGAWA
- An O(|*V*|3) algorithm for finding maximum flows in networks, V.M. Malhotra, M.Pramodh Kumar, S.N. Maheshwari
- Parallizing the Push-Relabel Max Flow Algorithm, Victoria Popic, Javier Vélez

NVIDIA