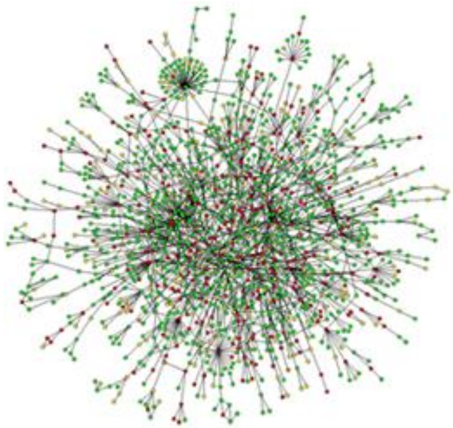
Optimizing Energy Consumption and Parallel Performance for Static and Dynamic Betweenness Centrality using GPUs

Adam McLaughlin, Jason Riedy, and David A. Bader

Georgia Tech College of Computing
Computational Science and Engineering

Motivation

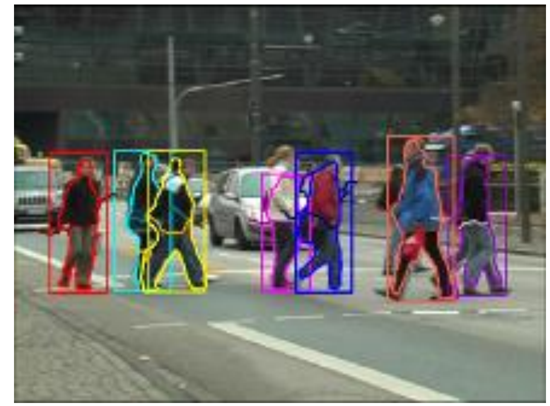
- Real world graphs are challenging to process
  - Enormous
    - Networks cannot be manually inspected
  - Varying structural properties
    - Small-world, scale-free, meshes, road networks
      - Not a one-size fits all problem
  - Unpredictable
    - Rapidly change over time
    - Data dependent memory access patterns



GRAPH 500

Motivation

- Graphs are no longer processed by supercomputers alone
  - Embedded systems
    - Computer vision
  - Mobile devices
    - Spam detection



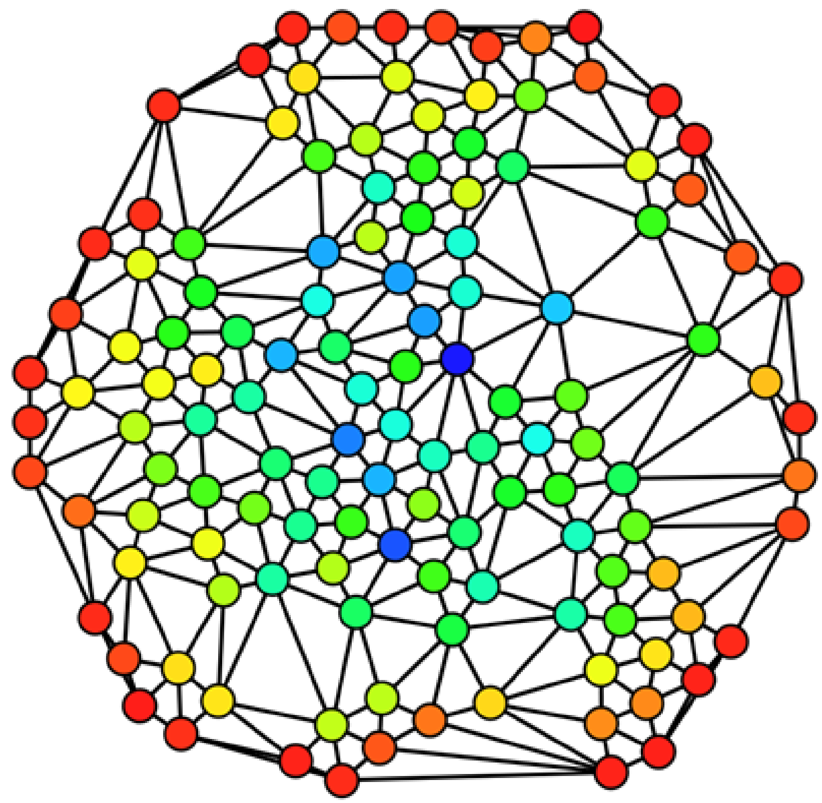
- Systems are becoming constrained by power and energy
  - High demand for work-efficient implementations
  - Goal: Maximize performance per Watt

Dynamic Analytics

- Update analytics rather than recompute them
  - Typically, a *local region* of the graph is affected
- A high throughput solution is desirable
  - Leverage the memory bandwidth of the GPU
  - Process each update in parallel
- A monumental task..
  - GPU kernels tend to be monolithic
  - Efficient parallel algorithms are lacking
  - Less intuitive to implement

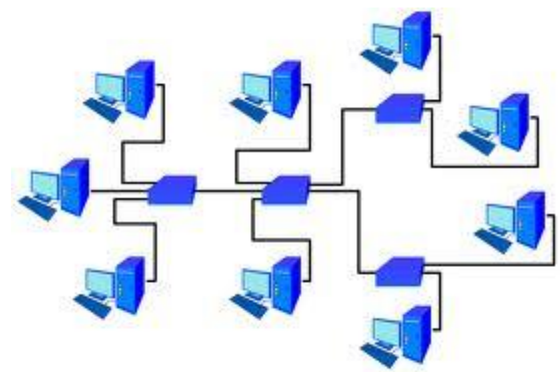
Betweenness Centrality

- Determine the importance of a vertex in a network
  - Requires the solution of the APSP problem
- Applications are manifold
- Computationally demanding
  - $O(mn)$ time complexity



Applications

- Deployment of detection devices in communication networks [Bye *et. al*]
- Analyzing brain networks [Rubinov and Sporns]
- Sexual networks and AIDS
- Identifying key actors in terrorist networks
- Transportation networks

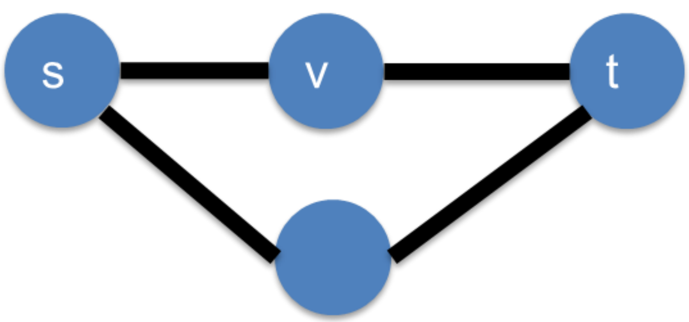


Defining Betweenness Centrality

- Formally, the BC score of a vertex is defined as:

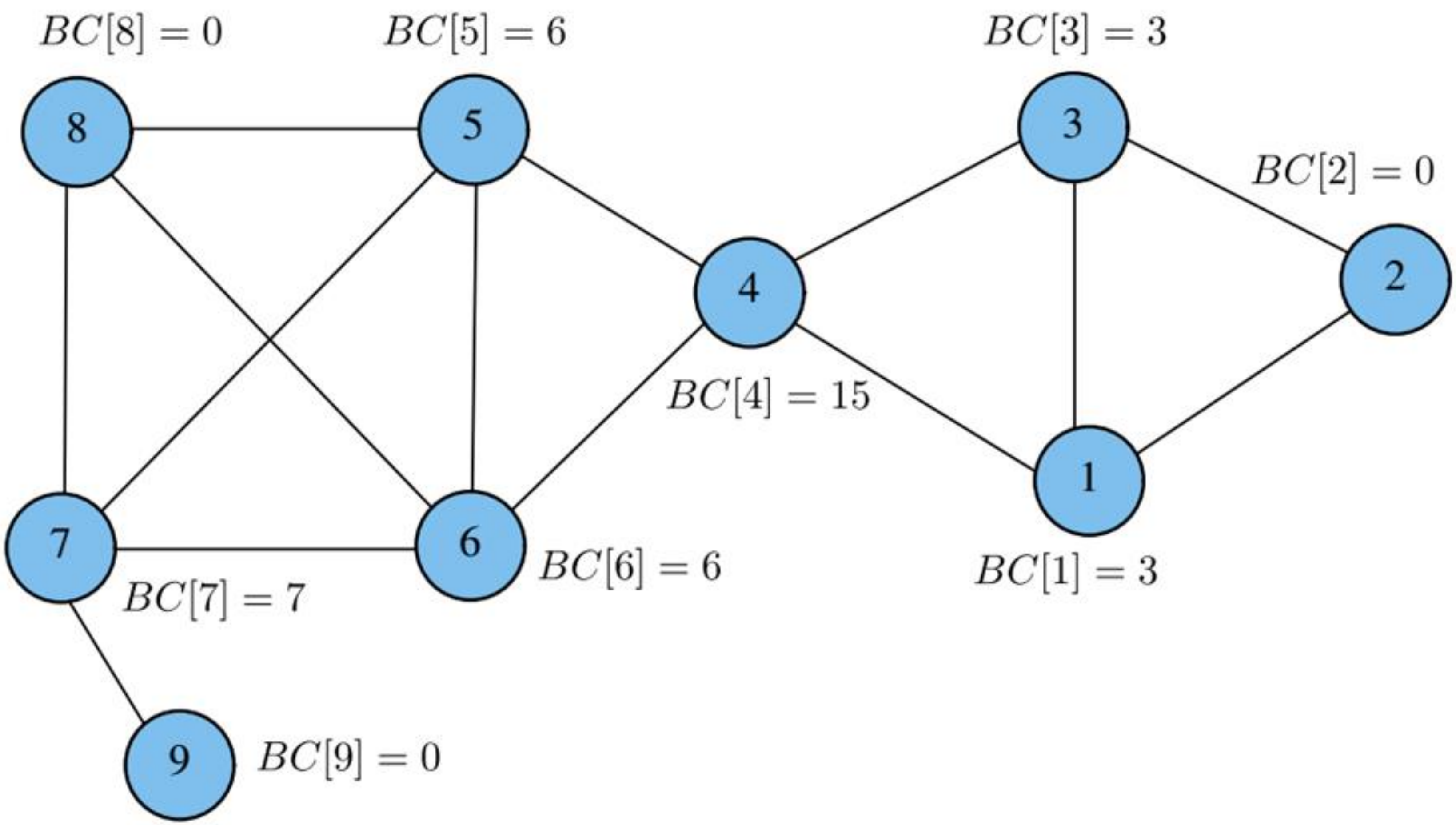
$$BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

  - σ_{st} is the number of shortest paths from s to t
  - $\sigma_{st}(v)$ is the number of those paths passing through v



$\sigma_{st} = 2$
$\sigma_{st}(v) = 1$

Example BC Calculation



Georgia Tech College of Computing

Brandes's Algorithm

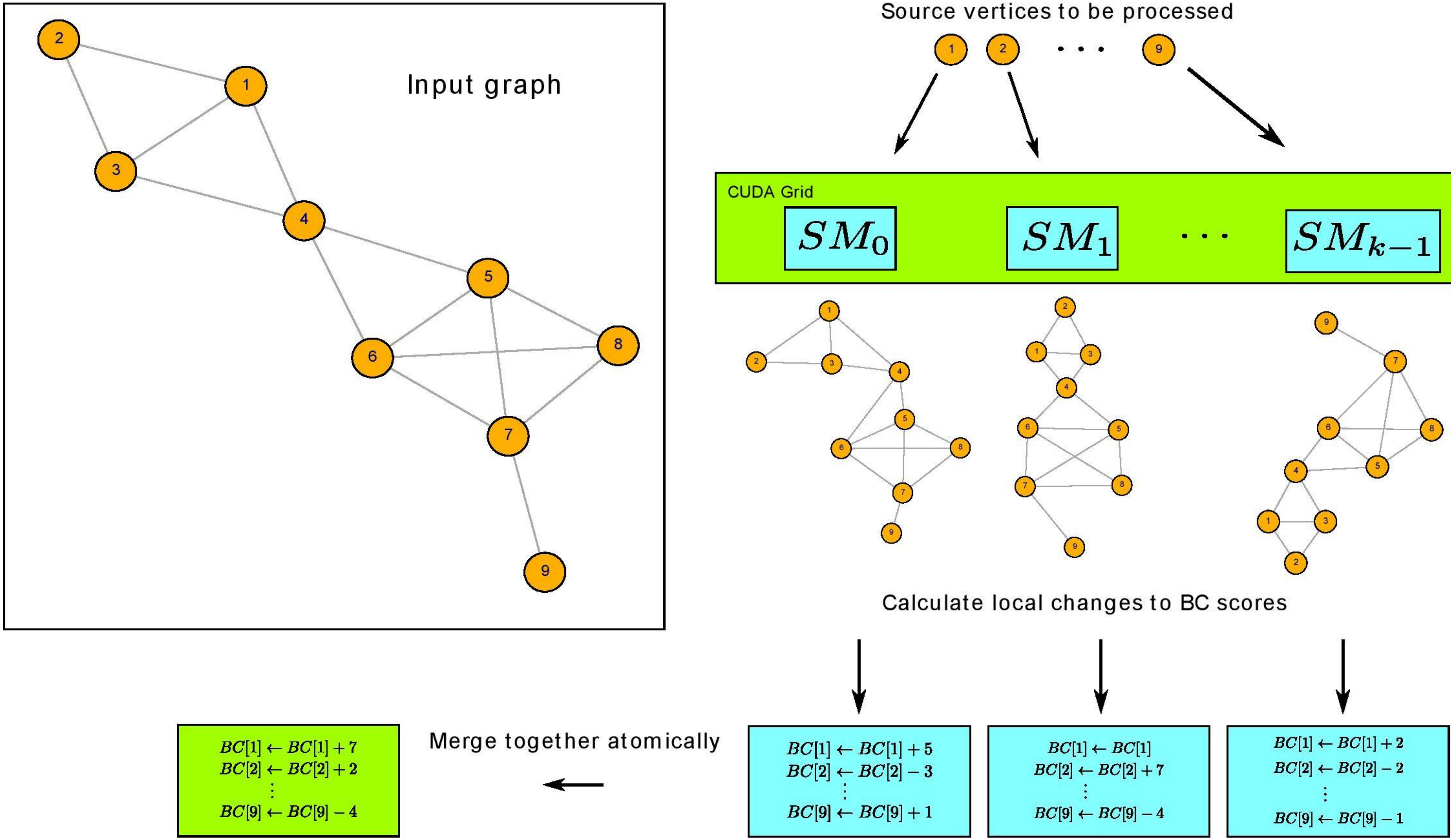
- Fastest known sequential algorithm
- Recursive relationship between BC scores contributed by a single vertex ("root")
  - Dependency:

$$\delta_s(v) = \sum_{w \in succ(v)} \frac{\sigma_{sv}}{\sigma_{sw}} \left(1 + \delta_s(w)\right)$$

  - Redefine BC scores as:

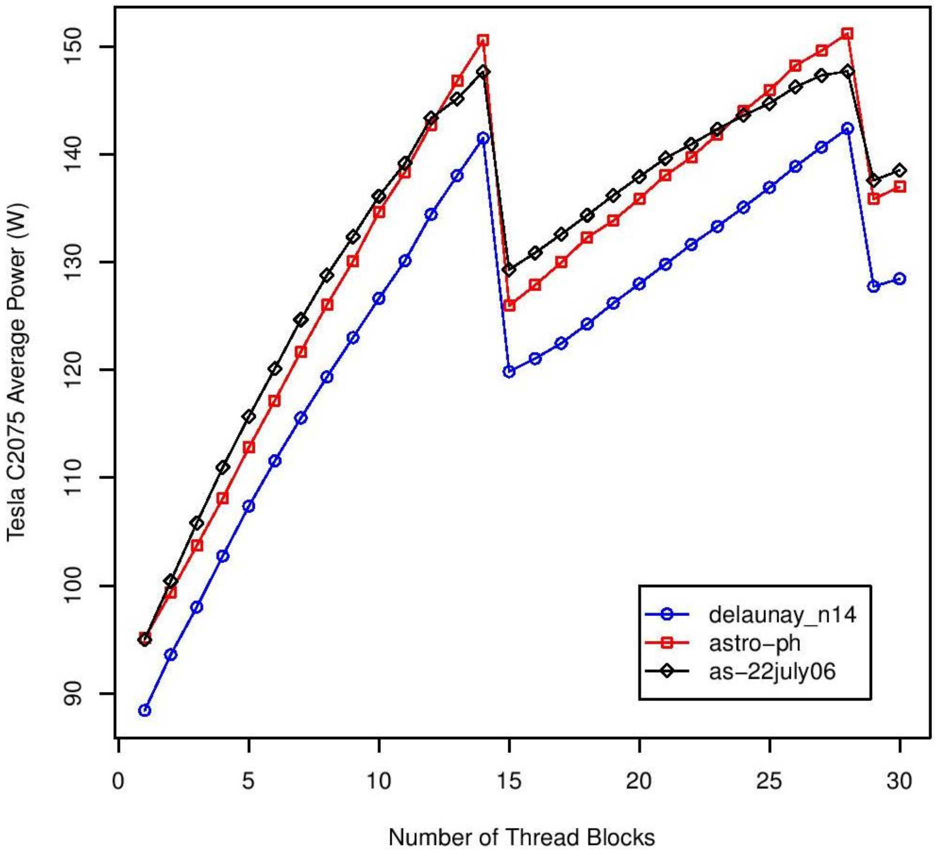
$$BC(v) = \sum_{s \neq \mathrm{v}} \delta_s(v)$$

Coarse-grained Parallelization Strategy



Power Consumption and Thread Blocks

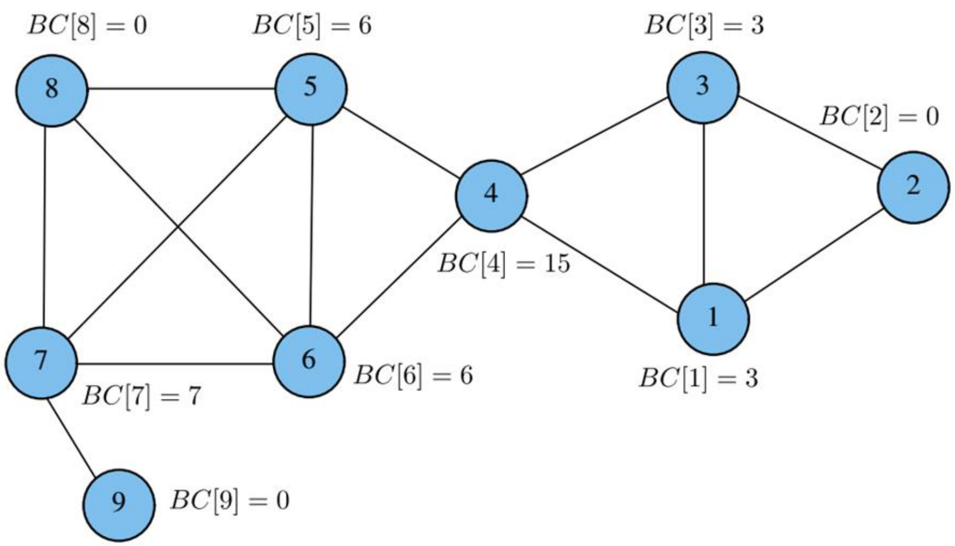
- HW does its best to idle SMs
- Number of thread blocks should be a multiple of the number of SMs
  – Performance scales linearly until all 14 SMs are busy

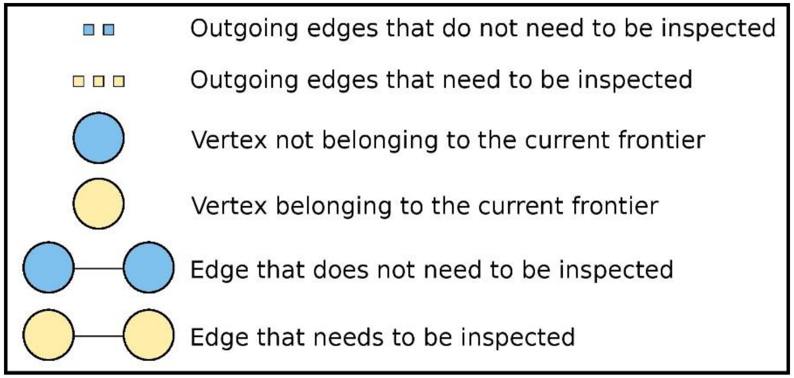


Prior GPU Implementations

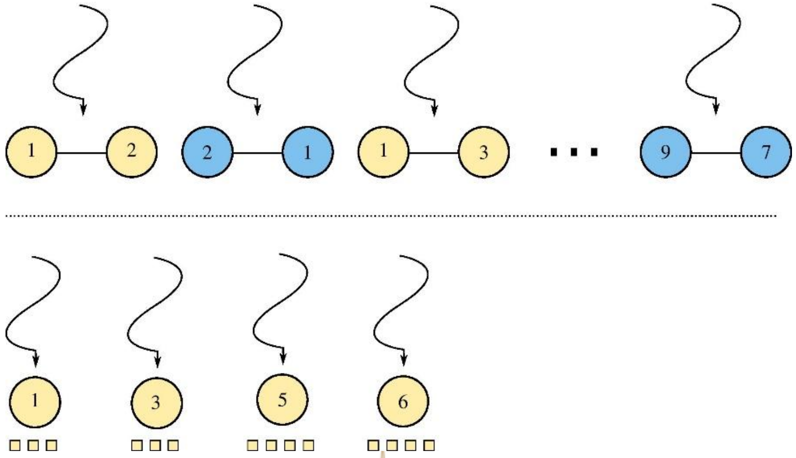
- Vertex and Edge Parallelism [Jia *et. al*]
  - Same coarse-grained strategy
  - Edge-parallel approach better utilizes the GPU
- GPU-FAN [Shi and Zhang]
  - Reported 11-19% speedup over Jia *et. al*
    - Results were limited in scope
  - Devote entire GPU to fine-grained parallelism
- Both use large predecessor arrays
- Both use $O(n^2 + m)$ graph traversals

Fine-grained Parallelization Strategy



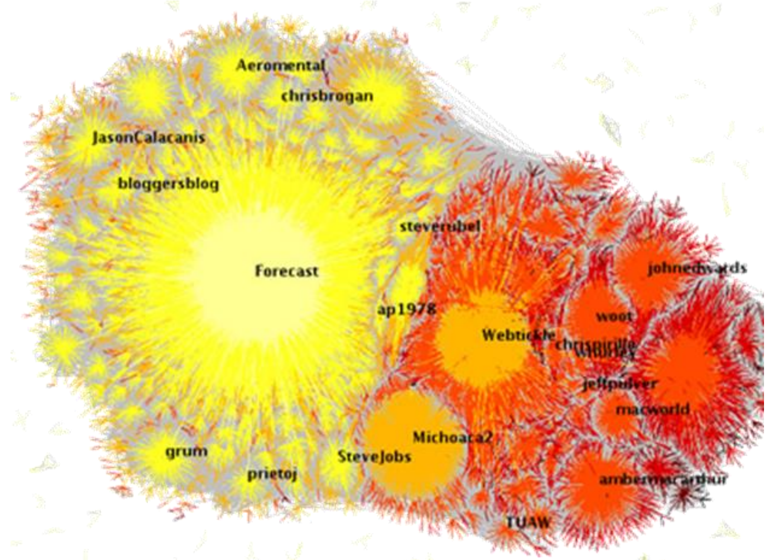


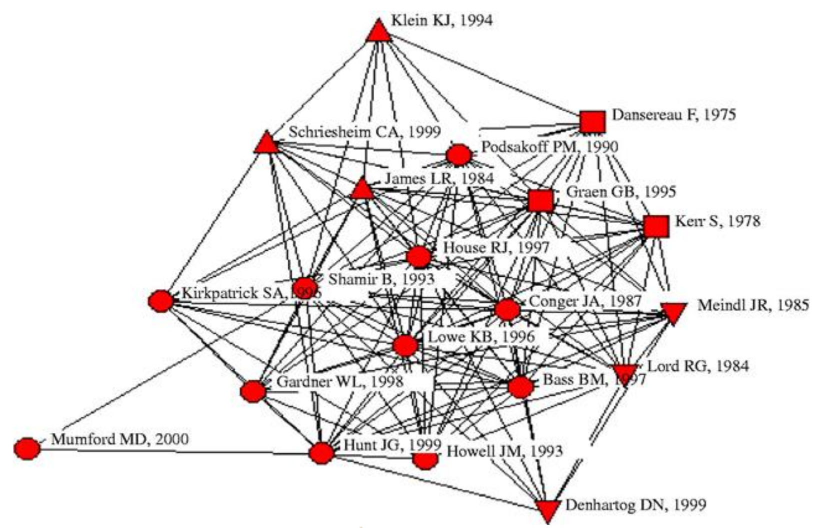
- Consider a BFS from vertex 4
- Expanding vertices {1,3,5,6}



Prior Dynamic Approaches

- Multiple implementations
  - – Sequential
  - – Resemble Green et al.
- Three update scenarios
  1. Same distance from the root
  2. Adjacent distances from root
  3. Greater than one level apart





Experimental Setup

GPU	SMs	Memory (GB)	Frequency (GHz)	Compute Capability	TDP (W)
Tesla C2075	14	6	1.15	2.0	225
Tesla K40c	15	12	0.745	3.5	245
GT 640 (Kayla)	2	1	0.95	3.5	75

- Kayla Platform
  - NVIDIA Tegra 3 ARM Cortex A9 CPU
    - 1.7 GHz single core
    - 32 KB L1 Instruction/Data Cache; 1 MB L2 Cache
    - 2 GB DDR3 RAM

Measuring Power

- Tesla GPUs
  - NVIDIA Management Library (NVML)
    - C-based API for measuring power
    - Sample at 10 ms intervals
- Kayla Platform
  - Watts Up wall-plug meter
  - Measures system power
    - CPU idle during GPU execution and vice versa
    - Sample at 1 ms intervals
- Power is averaged over the lifespan of a kernel

Benchmark Data Sets

Name	Vertices	Edges	Significance
delaunay_n12	4,096	12,264	Random Triangulation
delaunay_n20	1,048,576	3,145,686	Random Triangulation
kron_g500-logn16	55,321	2,456,071	Kronecker Graph
kron_g500-logn19	524,488	21,780,787	Kronecker Graph
luxembourg.osm	114,599	119,666	Road Network
preferentialAttachment	100,000	499,985	Scale-free
smallworld	100,000	499,998	Logarithmic Diameter

- Publicly available datasets
  - DIMACS 10^{th} Challenge; SNAP

Energy-efficiency of Static Calculations

- Define Traversed Edges per Second (TEPS):

$$TEPS_{BC}(G,t) = \frac{mn}{t}$$

Graph	Classification	Average Power (W)	MTEPS/W
delaunay_n20	Mesh	129.38	0.85
luxembourg.osm	Road Network	95.41	0.35
preferentialAttachment	Scale-free	127.18	1.33
smallworld	Logarithmic Diameter	127.10	2.54

- Low-diameter networks fully occupy the GPU
- Avg. Power is well below TDP (245 W)

Energy-efficiency of the embedded GPU

- CPU vs. GPU on the Kayla Platform (Dynamic)
- Times are averaged for 100 edge insertions

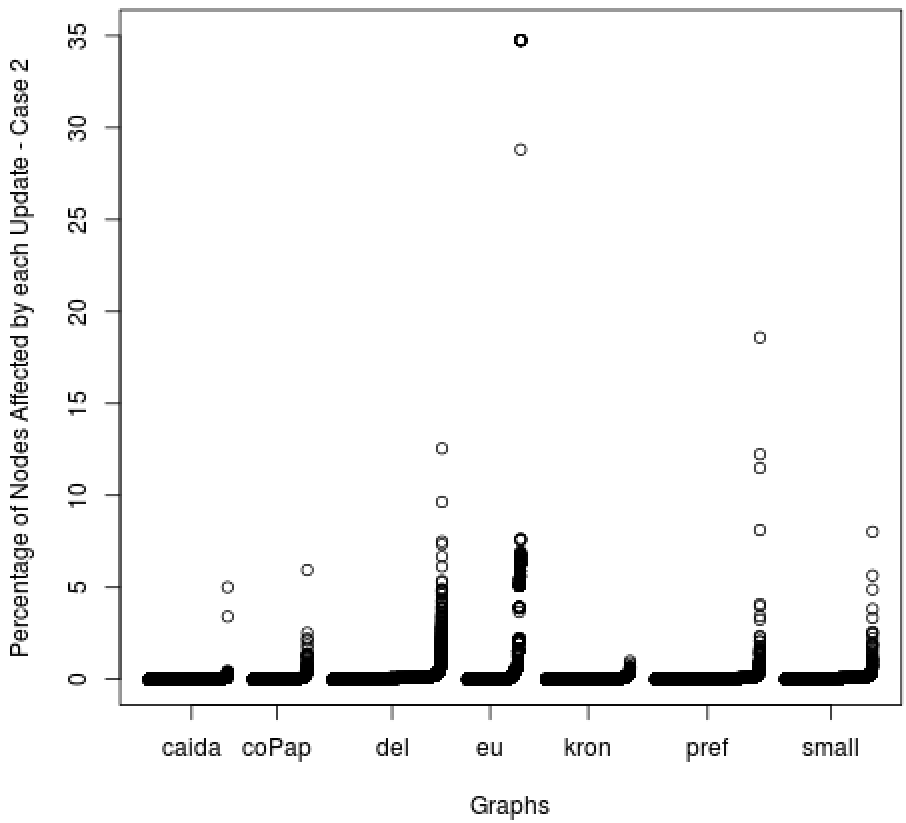
Graph	*delaunay_n12*	*kron_g500-logn16*
Solution Quality	Exact	Approximate ($k = 256$)
CPU Time (s)	35.44	33.79
GPU Time (s)	1.32	1.33
Speedup	**26.92x**	**25.39x**
Avg. CPU Energy (J)	914.35	875.08
Avg. GPU Energy (J)	42.64	43.79
Energy Savings	**95.3%**	**95.0%**
CPU MTEPS/W	0.05	0.72
GPU MTEPS/W	1.18	14.37

Energy-efficiency of Dynamic Calculations

- Static vs. Dynamic on the Kayla Platform (GPU)
- Times are averaged for 100 edge insertions

Graph	*delaunay_n12*	*kron_g500-logn16*
Solution Quality	Exact	Approximate ($k = 256$)
Static Time (s)	12.63	5.63
Dynamic Time (s)	1.32	1.33
Speedup	**9.6x**	**4.2x**
Static Energy (J)	424	188
Dynamic Energy (J)	42.6	43.8
Energy Savings	**90.0%**	**76.7%**
Static MTEPS/W	0.12	3.34
Dynamic MTEPS/W	1.18	14.37

Portion of graph affected by updates

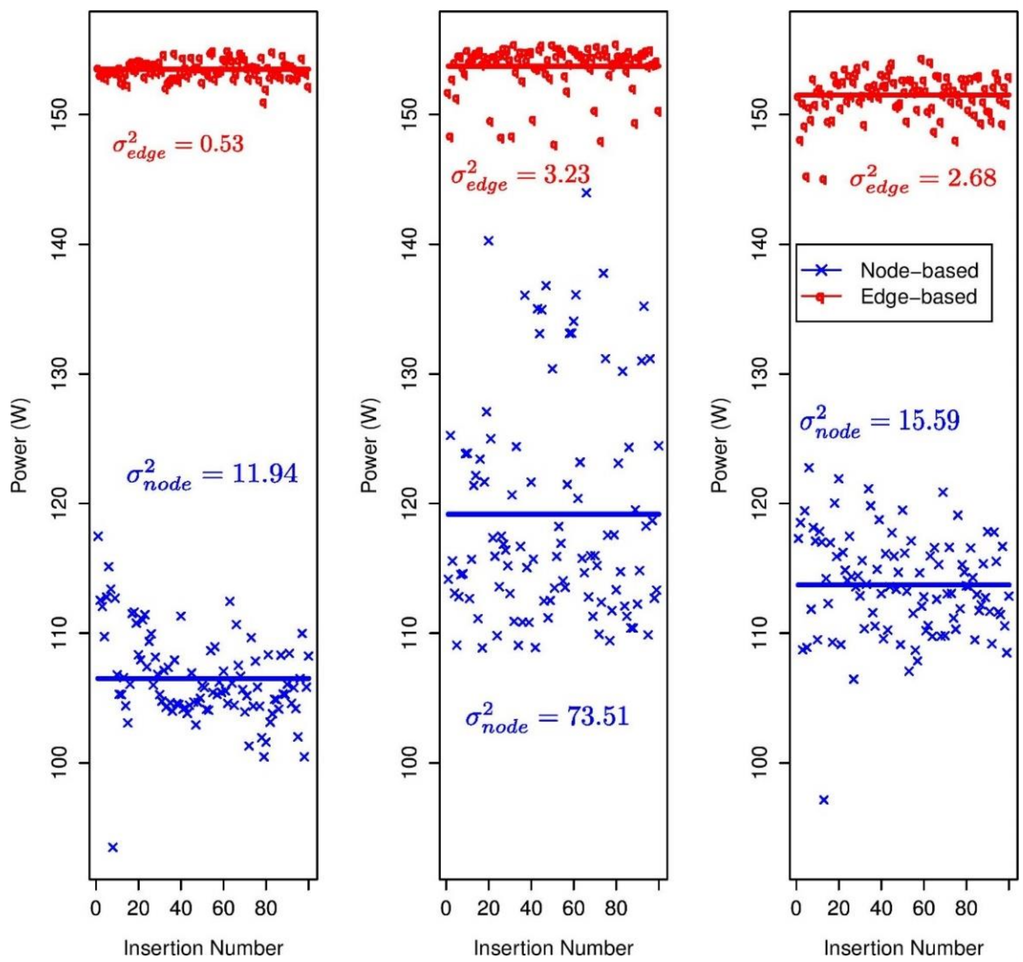


- 62,844 Adjacent insertions ("Case 2")
  - The worst insertion touched only ~35% of the nodes in the graph
  - Common insertion: Less than 1% of nodes touched

Georgia Tech College of Computing

Power Consumption by Traversal Method

- Edge-parallel method inspects all edges for all iterations
  - Consistent, wasteful work
- Work-efficient method requires considerably less power



Conclusions

- Energy reduction can be achieved through parallelism and dynamic algorithms
- Work-efficient algorithms are paramount
  - Updates tend to affect a local region of the graph
  - Better performance while using less power
  - Hybrid approaches for varying graph structures
- Programmability is a huge concern
  - Performance portability is difficult to obtain
    - Let library designers handle this burden

Acknowledgment of Support

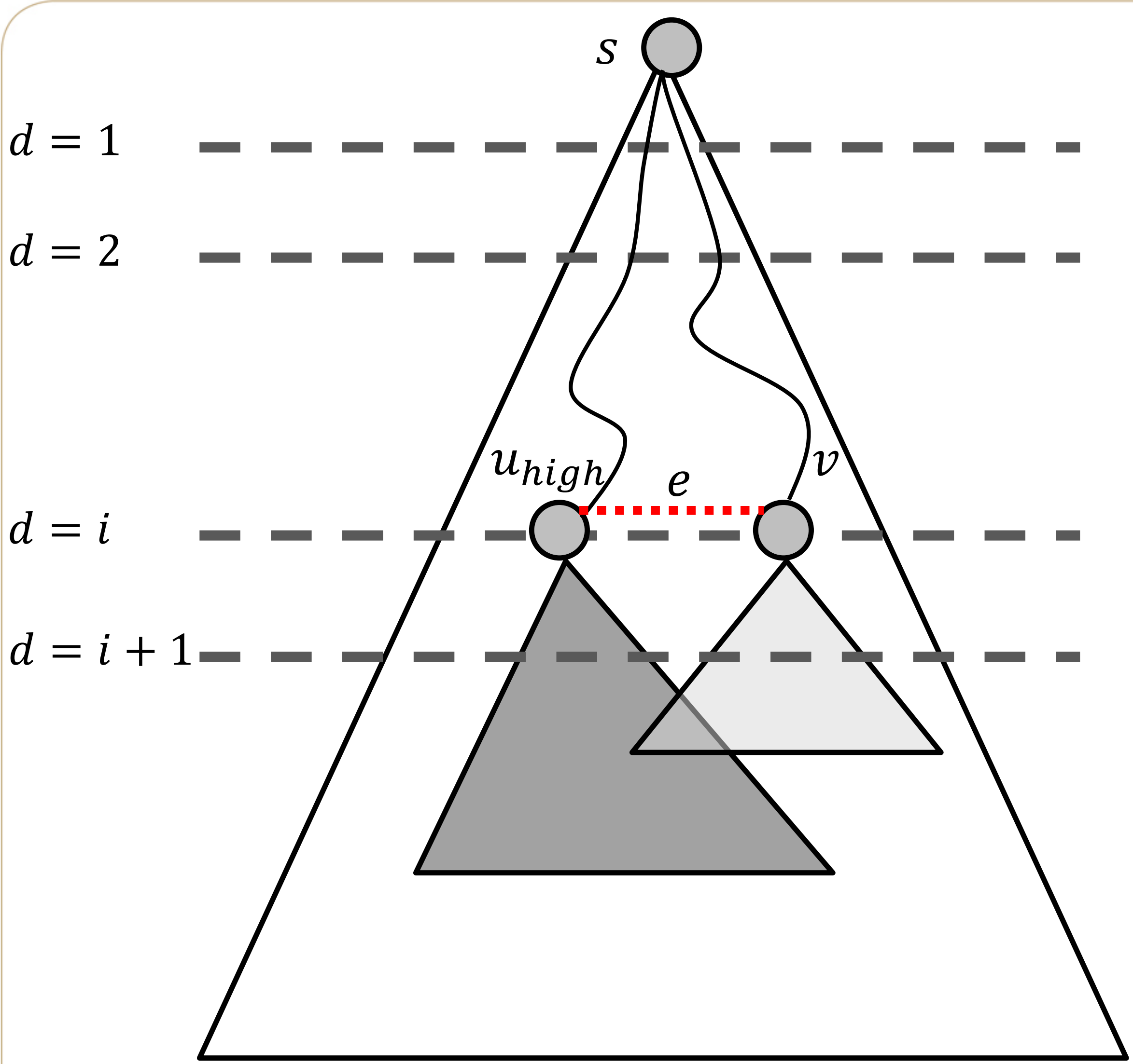
Questions

"To raise new questions, new possibilities, to regard old problems from a new angle, requires creative imagination and marks real advance in science."– Albert Einstein

https://github.com/Adam27X/hybrid_BC

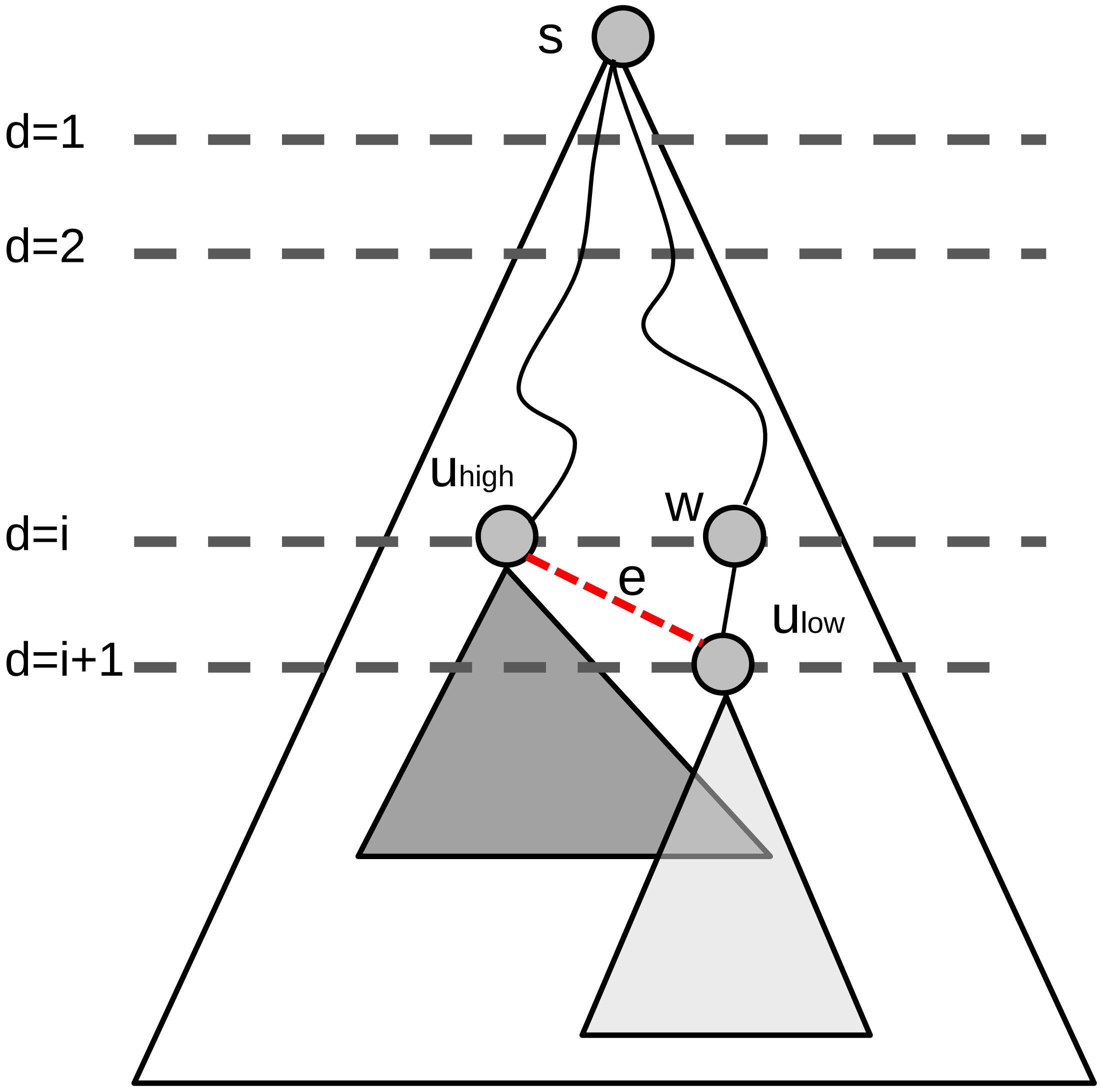
Backup

Case #1 – Same level



- New edge $e = (u, v)$
- No new shortest paths in this tree.

Case #2 – Adjacent levels

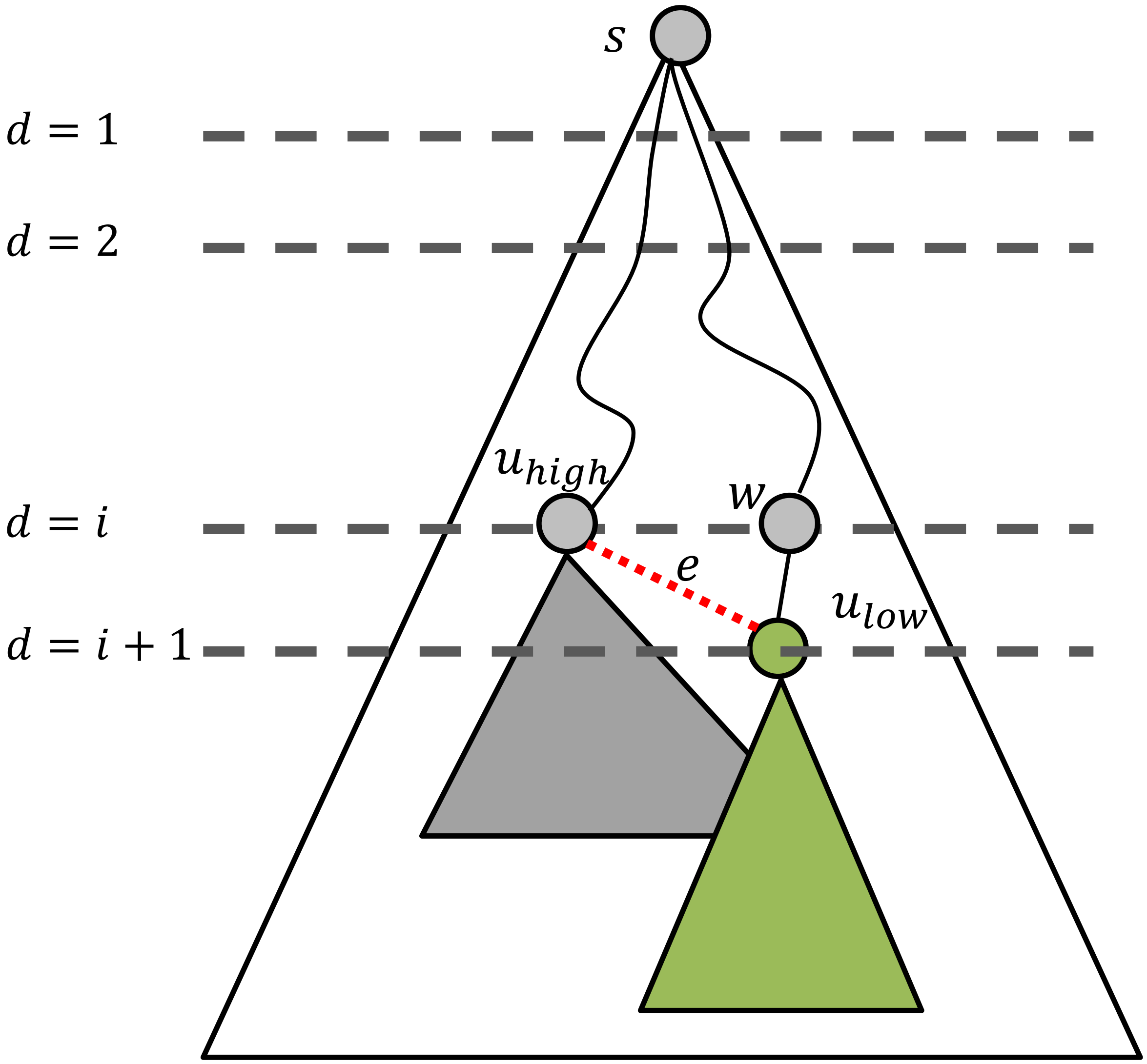


- New edge

  $e = (u_{high}, u_{low})$

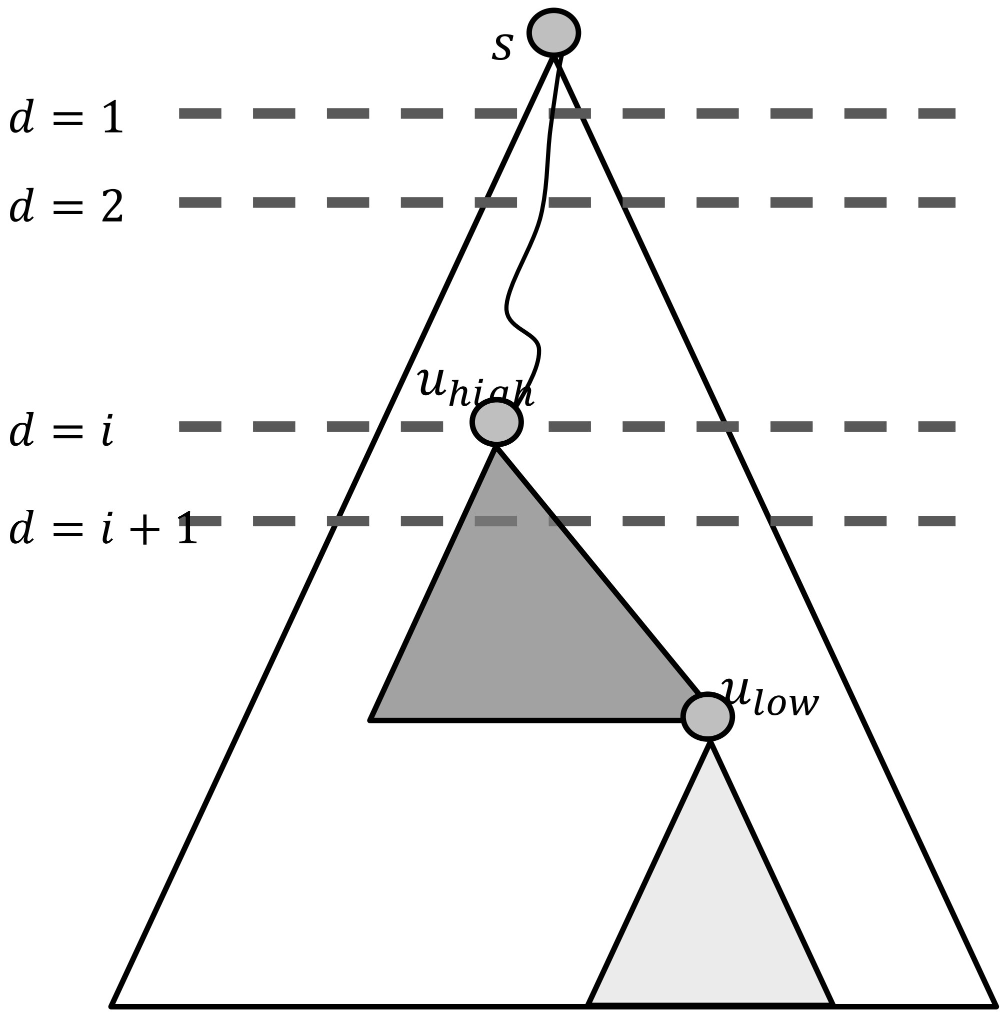
- All new paths go through e.

Case #2 – Adjacent levels



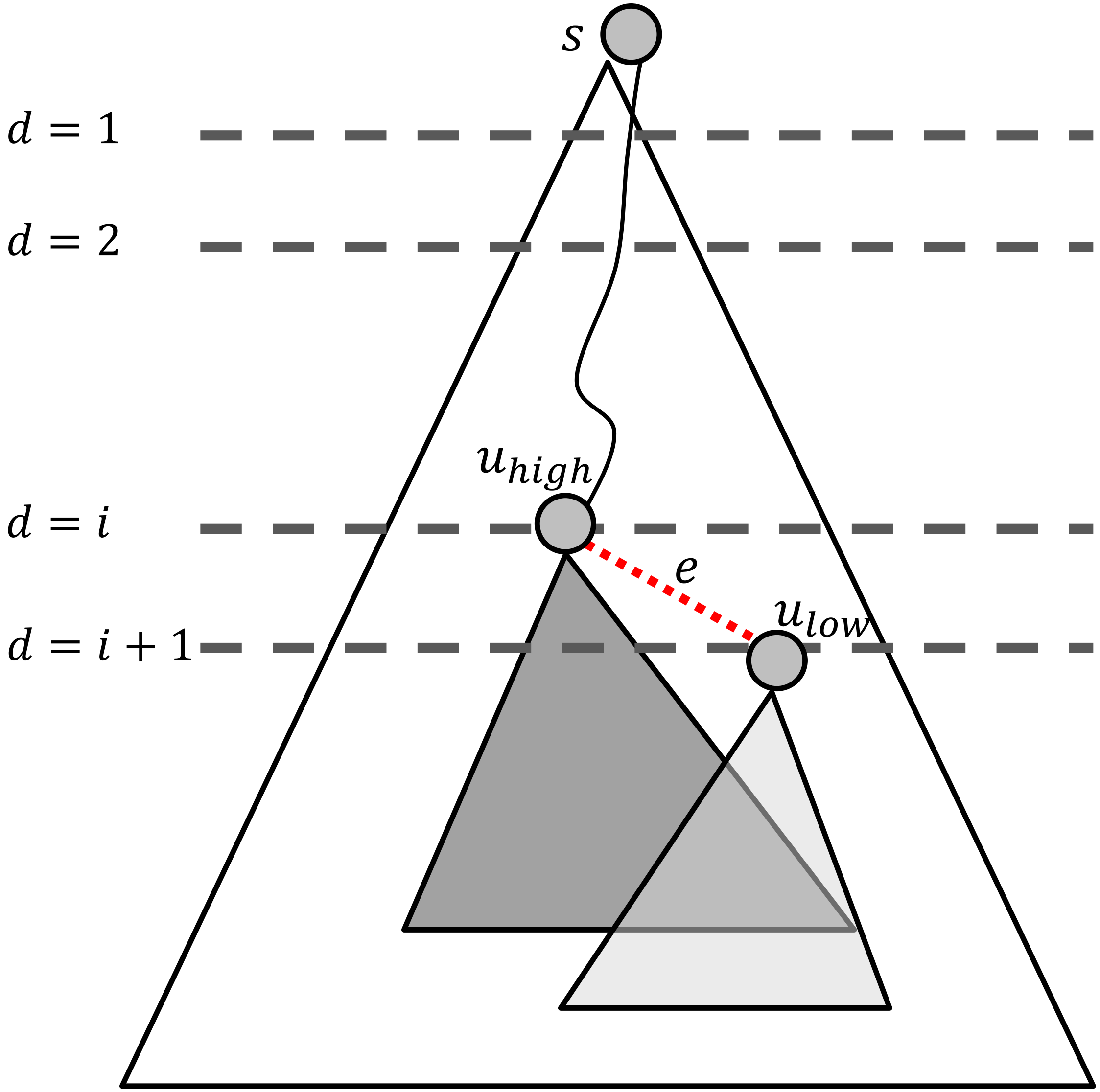
- No new shortest paths above u_{low}.
- Start BFS traversal at u_{low}.
- Fraction of edges/vertices traversed.

Case #3– Pull-up



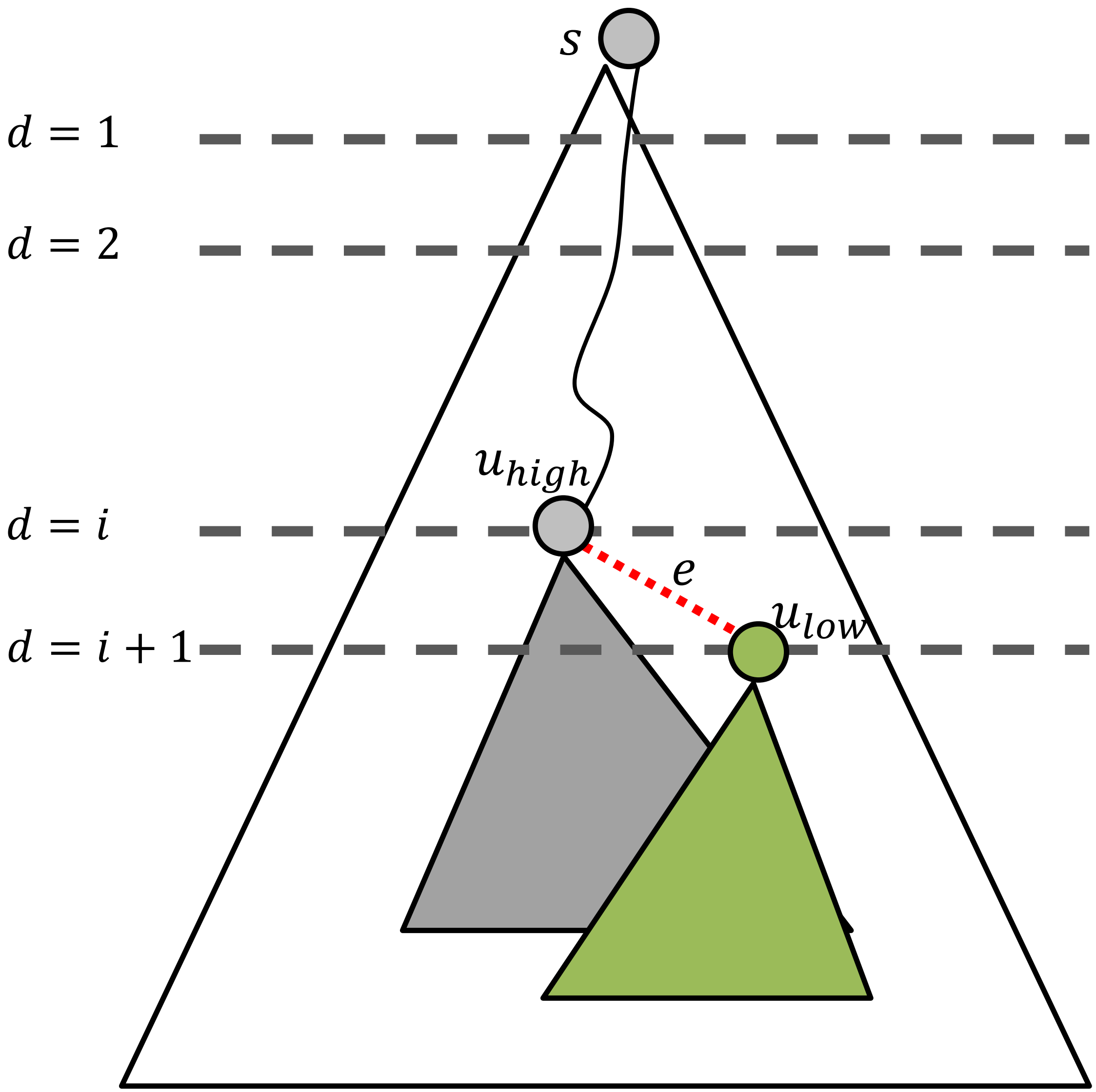
- New edge $e = (u_{high}, u_{low})$

Case #3– Pull-up



- New edge $e = (u_{high}, u_{low})$

Case #3– Pull-up



- No new shortest paths above u_{low}.
- Start BFS traversal at u_{low}.
- Fraction of edges/vertices traversed.