# Synthesizing Demonic Graph for Worst Case Performance

Mike He, Yihong Zhang

CSE 507 Presentation

# Motivations

- Given a program, measure worst case performance is hard
  - Algorithms that work well on random data may fail on specific forms of input data
  - It’s hard to construct input data that can expose program’s efficiency bottleneck
  - Heuristics make the problem worse
  - Data generated randomly always yield to certain distributions
- But it’s good if we can do so
  - Measure reliability of different heuristics
  - Automated data fuzzer for competitive programming events (ICPC, Codeforces, Google Code Jam, etc.)
  - Compare constant factors for various algorithms under worst case
- Let’s take SPFA as an example

# SPFA for Single Source Shortest Path (SSSP) Problem

$dist \leftarrow \text{MAKE-ARRAY}(n, [\infty]);$
$dist[s] \leftarrow 0;$
**for** $i \leftarrow 1$ *to* $|V| - 1$ **do**
    **for** $\{u, v, weight\} \in E$ **do**
        **if** $dist[u] + weight < dist[v]$ **then**
            $dist[v] \leftarrow dist[u] + weight;$
        **end**
    **end**
**end**

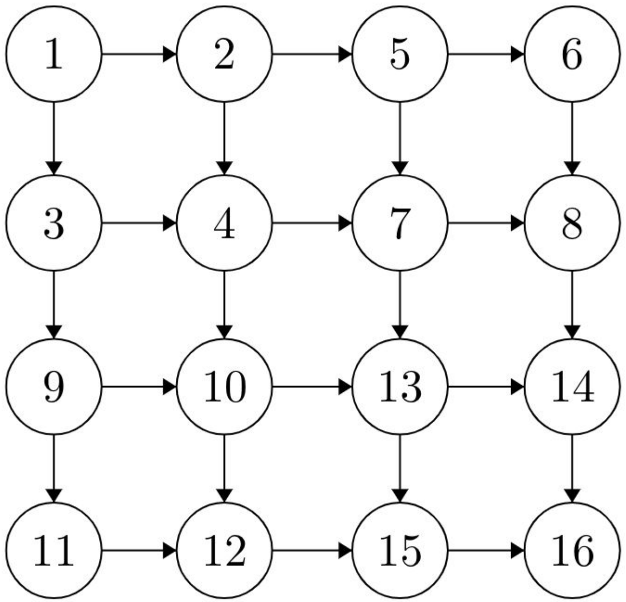
Bellman-Ford Algorithm

$Q \leftarrow \text{MAKE-QUEUE}();$
$dist \leftarrow \text{MAKE-ARRAY}(n, [\infty]);$
$dist[s] \leftarrow 0;$
Push $s$ into $Q$;
**while** $Q$ *is not empty* **do**
    $\mathbf{u} \leftarrow Q.\text{POP}();$
    **for** $\{\mathbf{u}, v, weight\} \in E$ **do**
        **if** $dist[\mathbf{u}] + weight < dist[v]$ **then**
            $dist[v] \leftarrow dist[\mathbf{u}] + weight;$
            Push $v$ into $Q$ if $v$ is not in $Q$;
        **end**
    **end**
**end**

SPFA

# How can we hack it?

- Construct a grid graph




Insight: this will create as many suboptimal paths as possible, and each suboptimal paths are expensive to SPFA.

# Heuristics!

- Let’s add some heuristics!
- Small Label First (SLF)
  - When pushing elements to the queue, check if the label (i.e., distance) on it is less than the one at the **front** of the queue. If so, push it at the front
- Large Label Last (LLL)
  - When pushing elements to the queue, check if the label (i.e., distance) on it is less than the **average value** of labels in the queue. If so, push it at the front
  - * this is actually a modified version of original LLL (at least the one on Wikipedia, which seems to be an counter-optimization)
- Will grid graphs still work?
  - No, they run in near linear time
- What other hacks we can think about?

# Sure...

- Construct a Shortest Path Tree (should be as long as possible)
  - Or simply a linked list
- Randomly add edges that connects two vertices with weights slightly greater than the path on the tree
- Shuffle the edges

Insight: this will also create many suboptimal paths, and each suboptimal paths have very similar distance to source, which will make the heuristics feel terrible…

But this SPT-based graph can no longer hack the original algorithm, and it still can’t hack SLF.

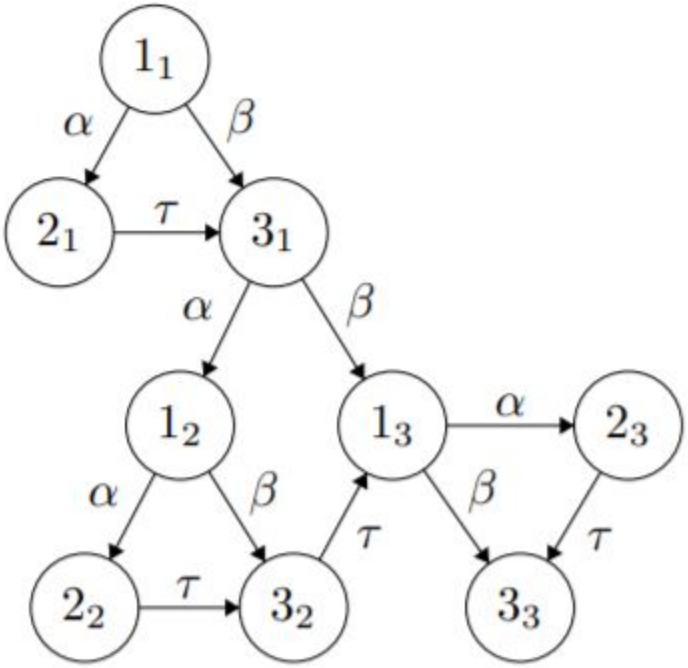
# Synthesizing Demonic Graph with SMT Solver

- Why not use a SMT solver?
- Pros
  - Entire process is automated
  - No longer need sophisticated human insights and empirical experiments
  - No longer need to construct graphs manually---sometimes such graph can be hard to constructed
- Cons
  - Can only synthesize a very small instance
    - We can scale it up!
  - Hard to determine what structural information is needed from the synthesized data
    - There is not an algorithm that is general enough to precisely capture the desired property of arbitrary graphs (More insight, more precision)

# Algorithm and Implementation

- Synthesize the gadget
  - Symmetry breaking on graph representation
  - Incremental solving
  - Symbolic-friendly optimization for various SPFA implementations
- Scale the gadget up
  - Specify an entry point and an exit point for the gadget
  - Everytime unroll a vertex into a gadget (self-similar)
  - Different exit point can cause Sager to simulate different structural information
    - We use two versions of start/exit: (start, end) and (start, start), where start and end are the first and last element pushed into the queue.

# Scaling: Connect By Exit Nodes




Scale Node 3

# Results - 100,000 Nodes & ~300,000 Edges

| Graphgen \ Algorithm | SPFA | SPFA+SLF | SPFA+LLL | Dijkstra |
|---|---|---|---|---|
| Random Graph | ≈0.47M | ≈0.5M | ≈0.6M | ≈0.4M |
| Grid | ≈230M | ≈0.5M | ≈0.5M | ≈0.4M |
| Linked List | ≈11M | ≈2.2M | ≈13M | ≈0.7M |
| Shortest Path Tree | ≈2.5M | ≈4M | **≈334M** | ≈0.4M |
| Sager | **≈9,778M** | **≈846M** | ≈0.8M | ≈0.7M |

# Results - 10,000 Nodes & ~30,000 Edges

| Graphgen \ Algorithm | SPFA | SPFA+SLF | SPFA+LLL |
|---|---|---|---|
| Random Graph | ≈45K | ≈48K | ≈58K |
| Grid | ≈1,500K | ≈52K | ≈49K |
| Linked List | ≈789K | ≈223K | ≈889K |
| Shortest Path Tree | ≈187K | ≈288K | **≈2M** |
| Sager | **≈97.8M** | **≈8.6M** | ≈79k |

# Limitations

- Sager can only hack short-sighted heuristics like SLF; it’s still very hard to synthesize a macro level property that can hack far-sighted heuristics like LLL
- There are some structural properties Sager can’t scale
  - E.g., can’t scale a grid graph from a square
  - But at least, if you have insight, Sager can help; If you don’t, Sager can provide a bottom line!