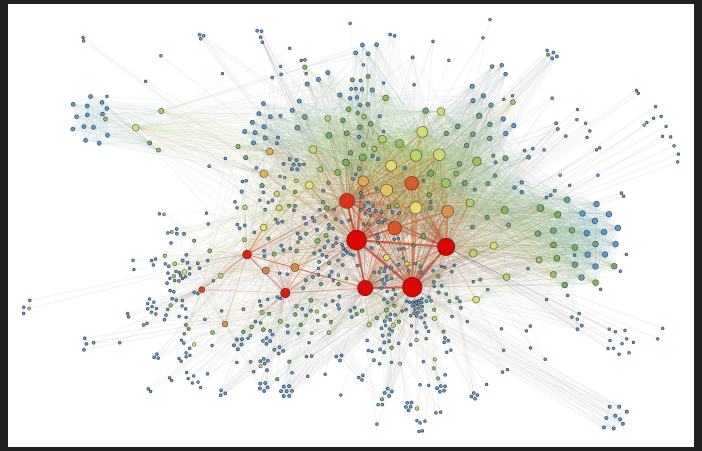
A Faster Algorithm for Betweenness Centrality

Ulrik Brandes

Presented by Siddhartha Jayanti

Problem

- Compute Node Centralities



$$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}$$ *closeness centrality* (Sabidussi, 1966)

$$C_G(v) = \frac{1}{\max_{t \in V} d_G(v,t)}$$ *graph centrality* (Hage and Harary, 1995)

$$C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$$ *stress centrality* (Shimbel, 1953)

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$ *betweenness centrality* (Freeman, 1977; Anthonisse, 1971)

Motivation

- Social Science: Find “popular” people
- Law & Order: Maximally disturb a criminal network
- Advertising: which entities will help a message spread?
- Counter-Intelligence: What communication nodes should be tapped?

Specific Problem of Interest

Compute “Betweenness Centrality” Quickly

σ_{st} = # shortest paths from s → t

$\sigma_{st}(v)$ = # shortest paths from s → t that include v

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

betweenness centrality
(Freeman, 1977; Anthonisse, 1971)

Bellman Criterion

Let $d_G(s, t)$ = distance from $s \rightarrow t$ in graph G

Lemma 1 (Bellman criterion) *A vertex $v \in V$ lies on a shortest path between vertices $s, t \in V$, if and only if $d_G(s,t) = d_G(s,v) + d_G(v,t)$.*

A simple observation

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$$

PAIR DEPENDENCY

$$\sigma_{st}(v) = \begin{cases} 0 & \text{if } d_G(s,t) < d_G(s,v) + d_G(v,t) \\ \sigma_{sv} \cdot \sigma_{vt} & \text{otherwise} \end{cases}$$

Previous Status Quo

Algorithmic Idea:

1. Compute all-pair shortest path, and number of paths
2. Sum all pair-dependencies $\delta_{st}(v)$

$$C_B(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v).$$

Previous Status Quo (Time-Space)

1. Can be done in $\Theta(n^3)$ time, $\Theta(n^2)$ space by modified Floyd-Warshall

2. Step two requires:

   $\Theta(n^2)$ storage (pair-dependency matrix)

   $\Theta(n^3)$ time (summation in step 2)

Goal

Improve Complexity to:

- O(n + m) space
- O(nm) time

Improvement for Sparse graphs ($m << n^2$)

IDEA #1: Use BFS / Dijkstra's

For shortest paths define *predecessors* of *v* from *s*:

$$P_s(v) = \{u \in V \,:\, \{u, v\} \in E,\, d_G(s, v) = d_G(s, u) + \omega(u, v)\}$$

Lemma 3 (Combinatorial shortest-path counting) *For* $s \neq v \in V$

$$\sigma_{sv} = \sum_{u \in P_s(v)} \sigma_{su}.$$

THEOREM

All shortest paths can be computed in:

$O(nm)$	time unweighted by augmented-BFS
$O(nm + n^2 \log n)$	time weighted by augmented-Dijkstra

⇒ Reduction in step (2) complexity improves full algorithm

IDEA #2: eliminate explicit sum

Define

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v)$$

Fast Implicit Sum Computation: FOR TREE

Lemma 5 *If there is exactly one shortest path from $s \in V$ to each $t \in V$, the dependency of s on any $v \in V$ obeys*

$$\delta_{s\bullet}(v) = \sum_{w\,:\,v \in P_s(w)} (1 + \delta_{s\bullet}(w)) .$$

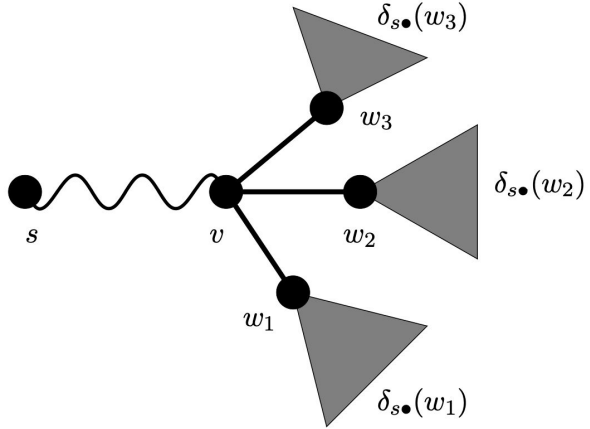


Figure 1: With the assumption of Lemma 5, a vertex lies on all shortest paths to its successors in the tree of shortest paths from the source

Fast Implicit Sum Computation

Theorem 6 *The dependency of $s \in V$ on any $v \in V$ obeys*

$$\delta_{s\bullet}(v) = \sum_{w\,:\,v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w)) \,.$$

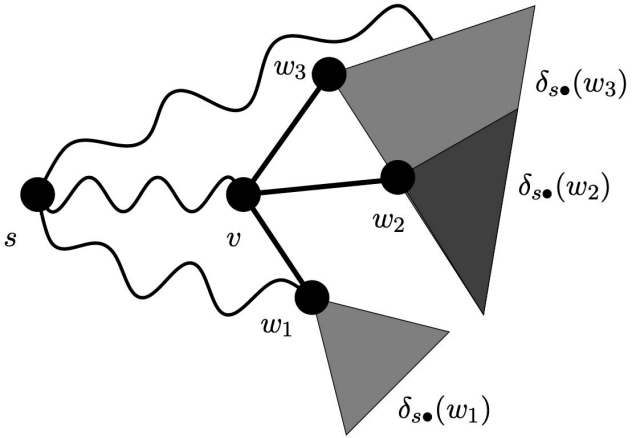


Figure 2: In the general case of Theorem 6, fractions of the dependencies on successors are propagated up along the edges of the directed acyclic graph of shortest paths from the source

THEORETICAL RESULT

Corollary 7 *Given the directed acyclic graph of shortest paths from $s \in V$ in G, the dependencies of s on all other vertices can be computed in $\mathcal{O}(m)$ time and $\mathcal{O}(n+m)$ space.*

Theorem 8 *Betweenness centrality can be computed in $\mathcal{O}(nm + n^2 \log n)$ time and $\mathcal{O}(n+m)$ space for weighted graphs. For unweighted graphs, running time reduces to $\mathcal{O}(nm)$.*

EXPERIMENT: Computation for Large Dense Graphs

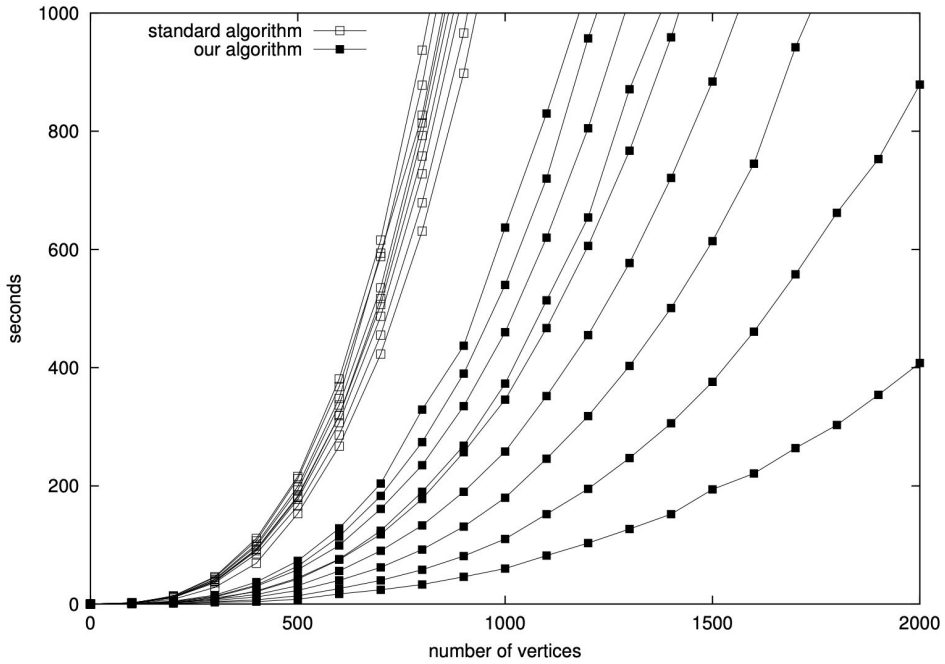


Figure 3: Seconds needed to the compute betweenness centrality index for random undirected, unweighted graphs with 100 to 2000 vertices and densities ranging from 10% to 90%

EXPERIMENT: Speed for Constant Degree Graph

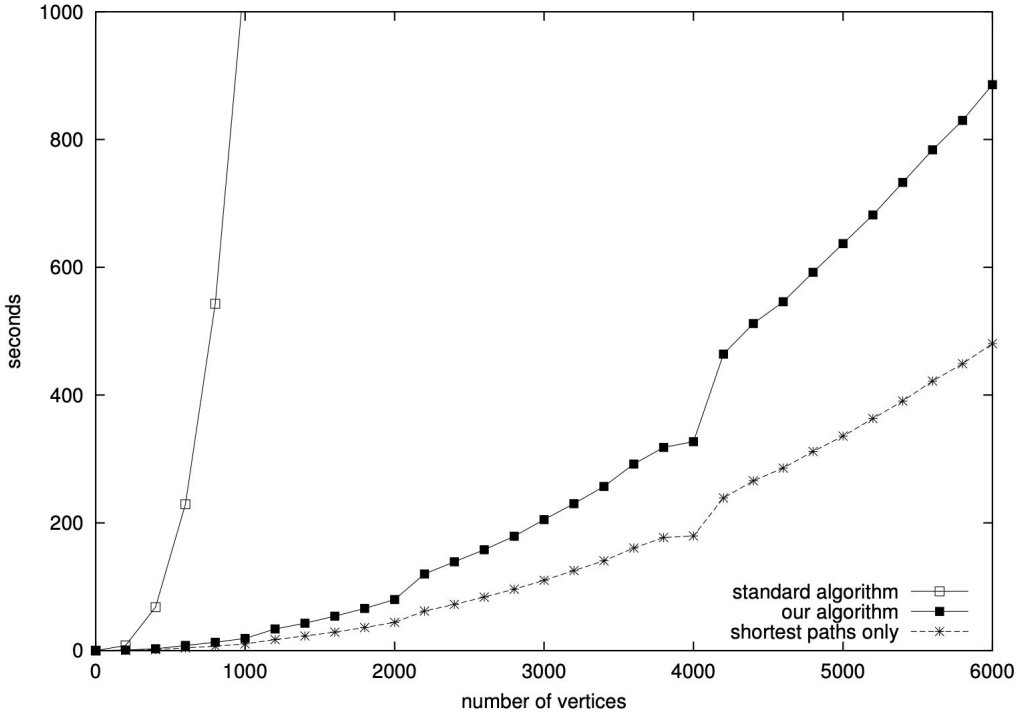


Figure 4: Seconds needed to the compute betweenness centrality index for random undirected, unweighted graphs with constant average degree 20. The funny jumps are attributed to LEDA internals

Strengths and Weaknesses of Paper

Strengths:

+ Theoretical Improvement
+ Betweenness-Centrality is computable for larger graphs (empirically)
+ Broad approach helps computation times for other centrality measures

Weakness:

- Doesn’t make use of parallelism

Discussion Questions

- Can we exploit parallelism and these techniques together?

- How important is the centrality measure? Some of the other measures are cheaper to compute.

- The graphs in the paper are much smaller than modern Social-Networks; what can we do?