Main Memory Evaluation of Recursive Queries on Multicore Machines

Mohan Yang, Carlo Zaniolo

UCLA CS Department

SCALABLE ANALYTICS INSTITUTE

Motivation

- Big data analytics on parallel systems like multicore machines and multi-node clusters.
- Many advanced applications require iteration and/or recursion.
  - PageRank, transitive closure, community discovery, network centrality, etc.
- But how to implement recursion on these new massively parallel systems is not well understood.

Motivation

- Recent studies [Afrati et al., 2011], [Afrati et al., 2012], [Shaw et al., 2012] on implementing transitive closure in multi-node clusters.
- Algorithms that deliver optimal performance on multi-node clusters are hardly optimal on multicore machines.
- Thus, we study the evaluation of transitive closure query on multicore machines.

Related Work

- Semi-naive evaluation, Smart, Floyd-Warshall
- I/O efficient algorithms
    - [Warren et al., 1975] [Agrawal et al., 1987] …
- Parallel algorithms
    - [Valduriez et al., 1988] [Agrawal et al., 1988] [Wolfson et al., 1992] [Cacace et al., 1993]

Transitive Closure (TC)

Consider a relation $\mathrm{arc}(X, Y)$ that represents the edges of a directed graph, the *transitive closure* of arc is a relation $\mathrm{tc}(X, Y)$ such that it contains all pairs (X, Y) that X can reach Y via a path in the graph.

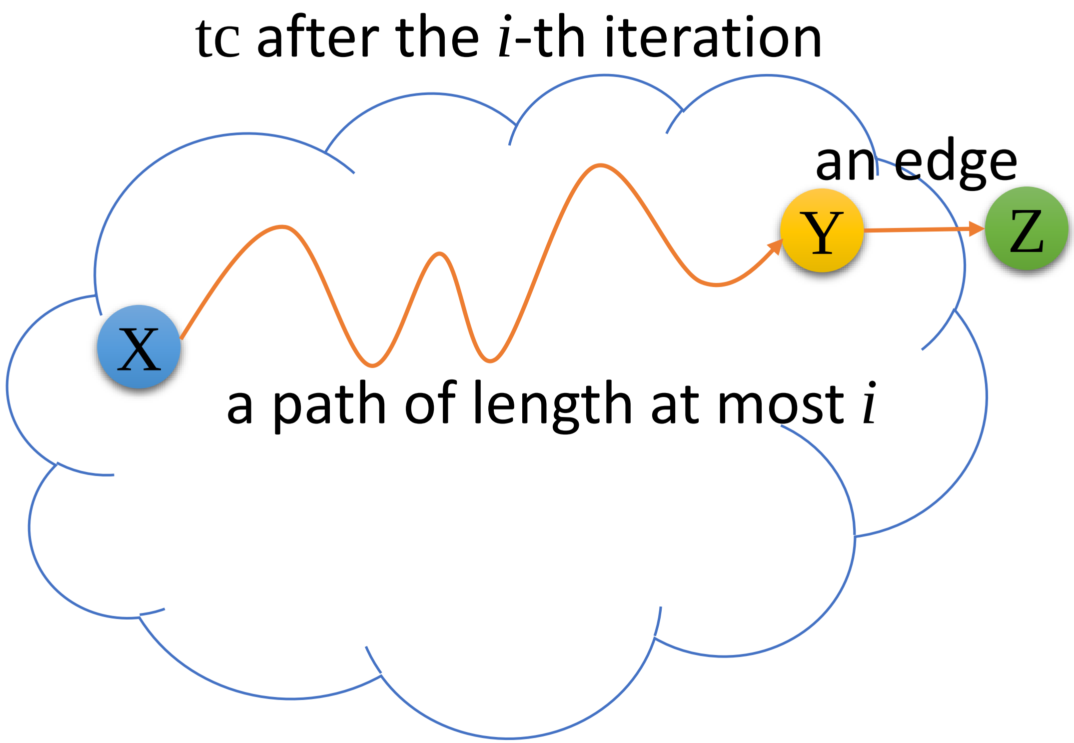
$\mathrm{tc}(X, X)$ <- $\mathrm{arc}(X, \_)$.

$\mathrm{tc}(X, Y)$ <- $\mathrm{tc}(X, Z), \mathrm{arc}(Z, Y)$.

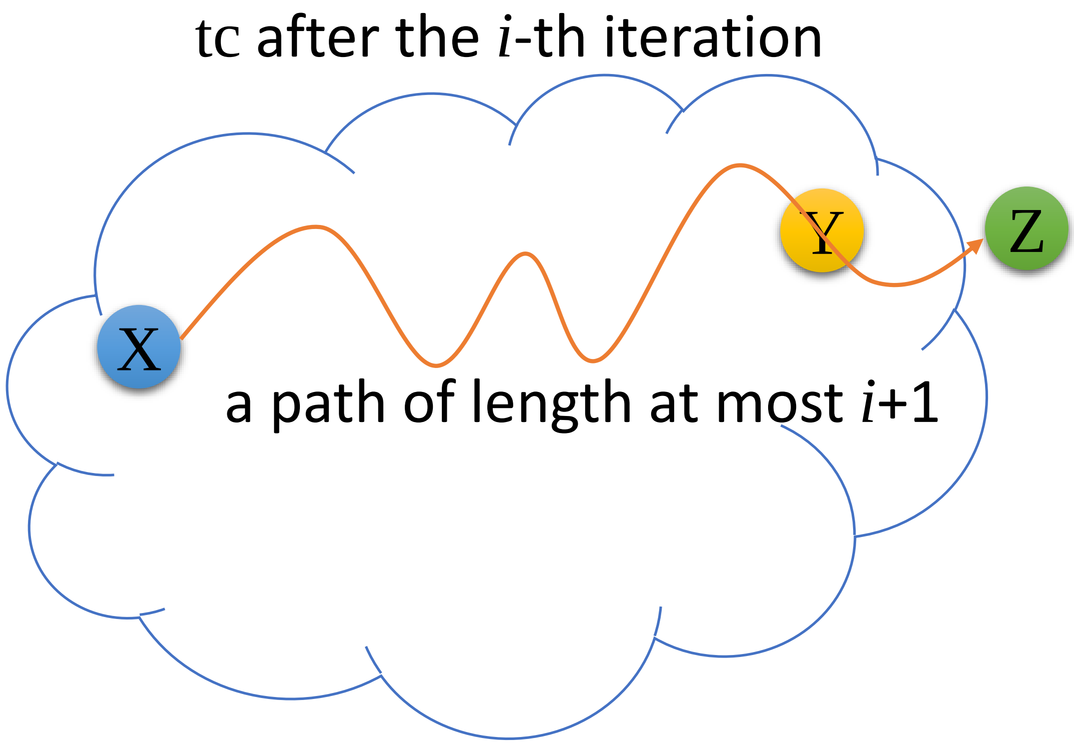
Similar problems: all-pairs shortest path, bill of materials, maximum capacity path, critical path, most reliable path, etc.

Evaluation Algorithms

Semi-Naive Evaluation



Semi-Naive Evaluation

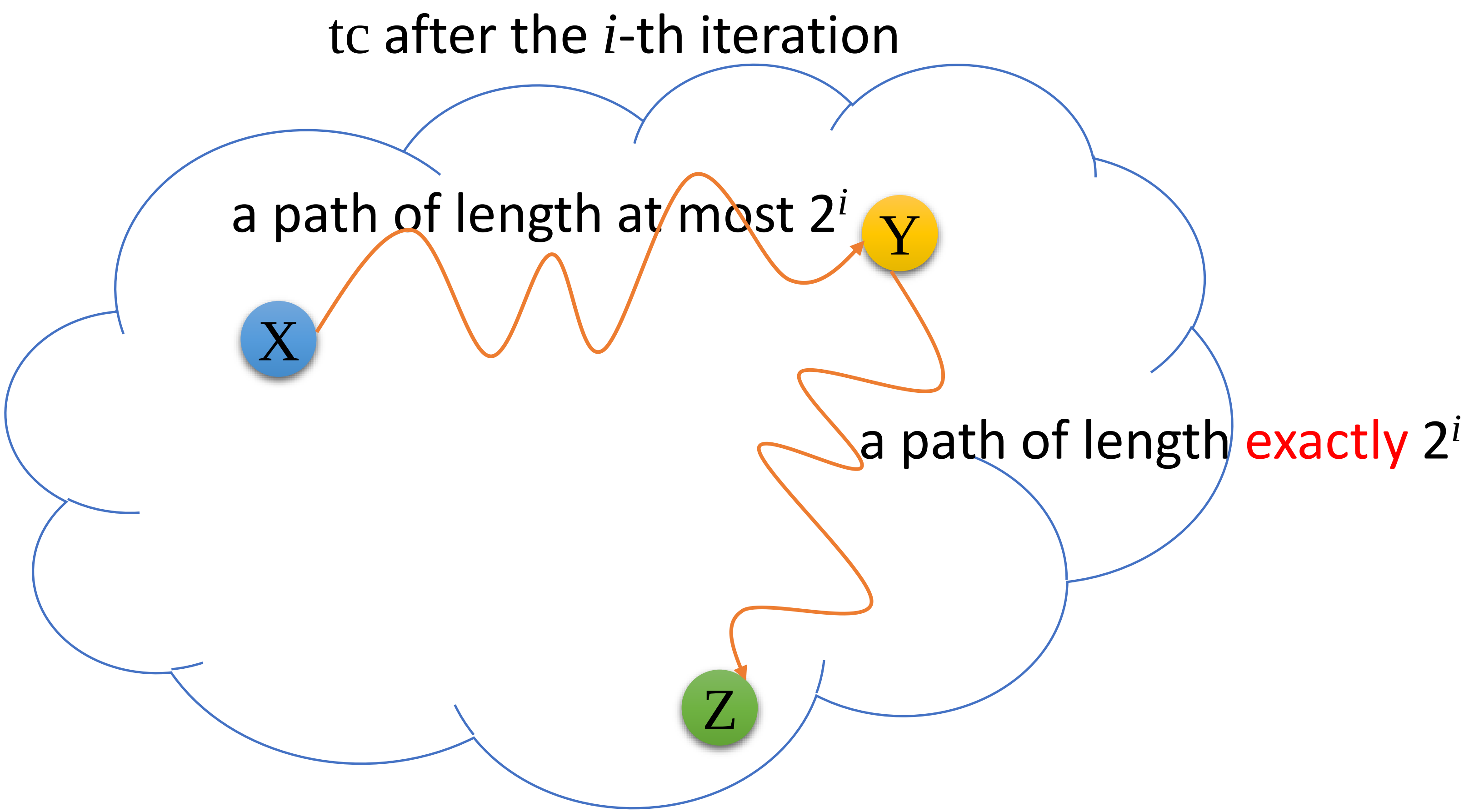


Semi-Naive Evaluation

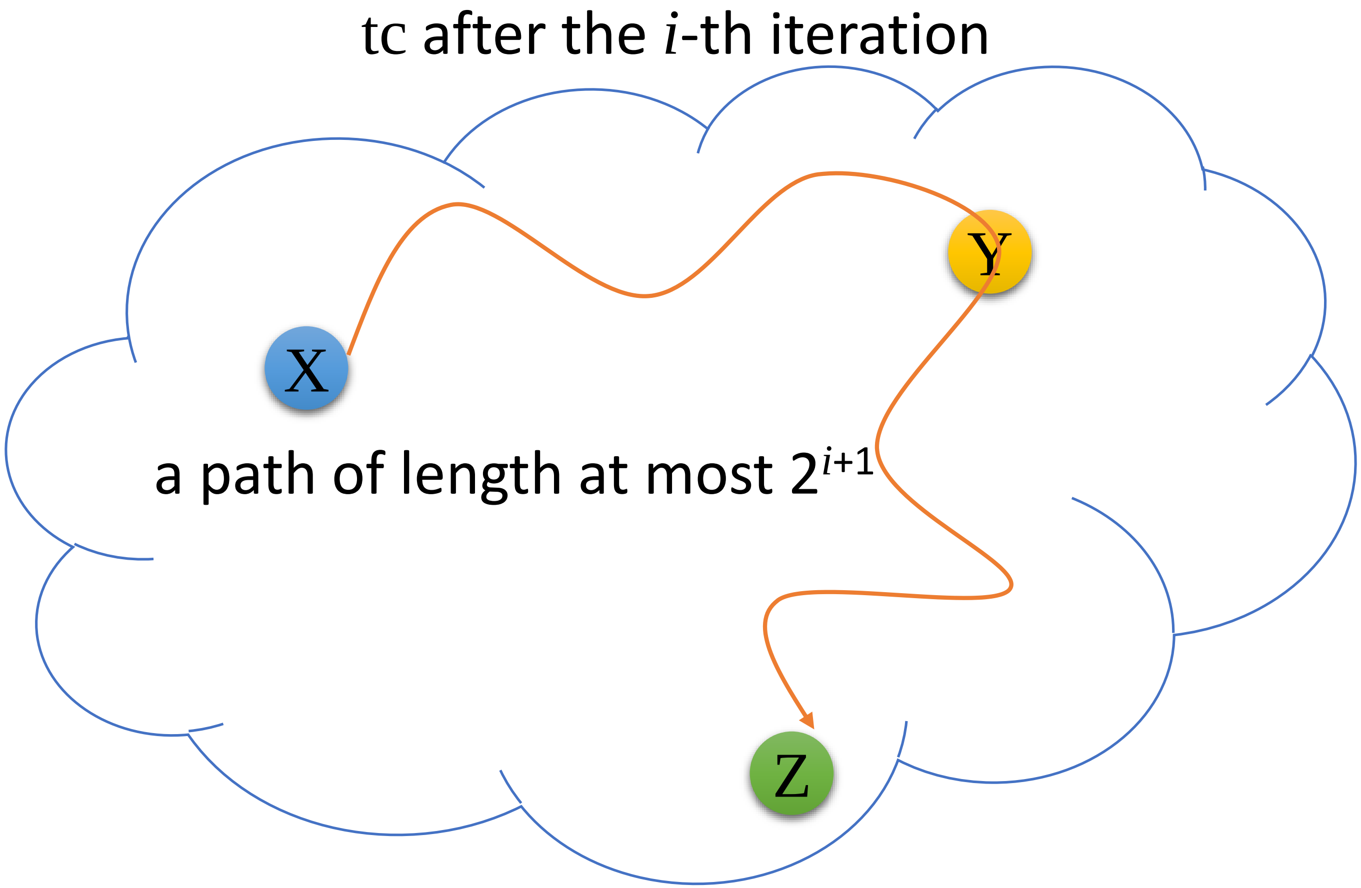
$$\mathrm{tc}_0(\mathrm{X},\mathrm{X}) = \mathrm{arc}(\mathrm{X},\_), \mathrm{tc} = \mathrm{arc}(\mathrm{X},\_)$$
$$\mathrm{tc}_{i+1} = \pi_{\mathrm{X,Z}}\mathrm{tc}_i(\mathrm{X},\mathrm{Y}) \bowtie \mathrm{arc}(\mathrm{Z},\mathrm{Y}) - \mathrm{tc}$$
$$\mathrm{tc} = \mathrm{tc} \cup \mathrm{tc}_{i+1}$$

- Require O(n) iterations for an n-vertex graph.
- Build an index (adjacency list) on arc since it is unchanged between iterations.

Smart Algorithm



Smart Algorithm



Smart Algorithm

$$\mathrm{tc}_0(\mathrm{X}, \mathrm{X}, 0) = \mathrm{arc}(\mathrm{X}, \_)$$
$$\mathrm{tc}_0(\mathrm{X}, \mathrm{Y}, 1) = \mathrm{arc}(\mathrm{X}, \mathrm{Y})$$
$$\mathrm{tc}_{i+1} = \mathrm{tc}_i \cup \pi_{\mathrm{X,Z,L}+2^i} \mathrm{tc}_i(\mathrm{X}, \mathrm{Y}, \mathrm{L}) \bowtie \mathrm{tc}_i\left(\mathrm{Y}, \mathrm{Z}, 2^i\right)$$

- No redundant derivations.
- Require O(log n) iterations for an n-vertex graph.

Single-Source Closure Algorithms

tc(X, X) <- arc(X, _).
tc(X, Y) <- tc(X, Z), arc(Z, Y).

Instantiate variable X with a constant x where x a vertex in the graph.

tc(x, x) <- arc(x, _).
tc(x, Y) <- tc(x, Z), arc(Z, Y).

For each vertex x, use semi-naive to compute its *single-source closure* (SSC).

- SSC1 – represent SSC as a set.
- SSC2 – represent SSC as an array.

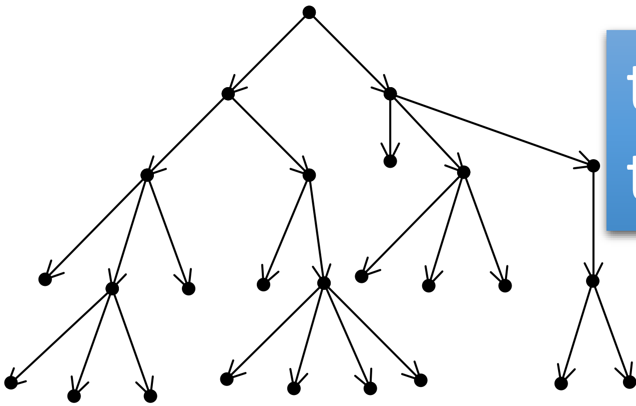
Algorithm Implementations

- Parallel implementation in C++
  - Semi-naive
  - Smart

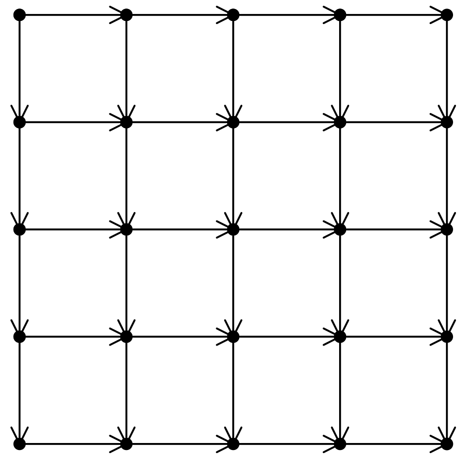
  In memory parallel hash join/set difference/union.
  - SSC: each thread works on one vertex at a time.
- Testing environment
  - A multicore machine with 4 AMD CPUs (64 cores in total) and 256G memory (8 NUMA regions).
- An optimization on NUMA hardware
  - Duplicate arc (adjacency list) on each NUMA region.

Performance Comparison

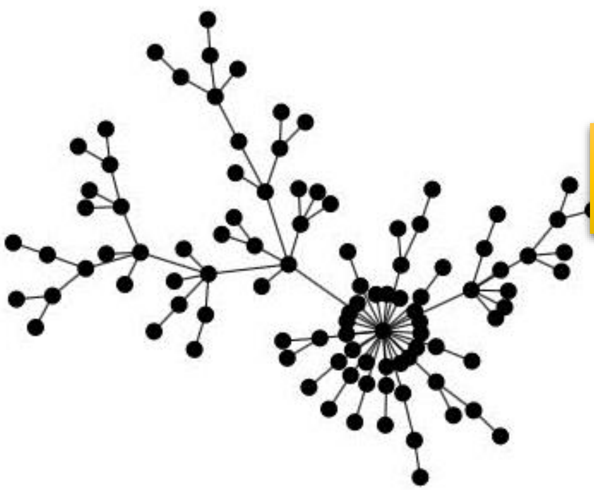
Synthetic Test Graphs



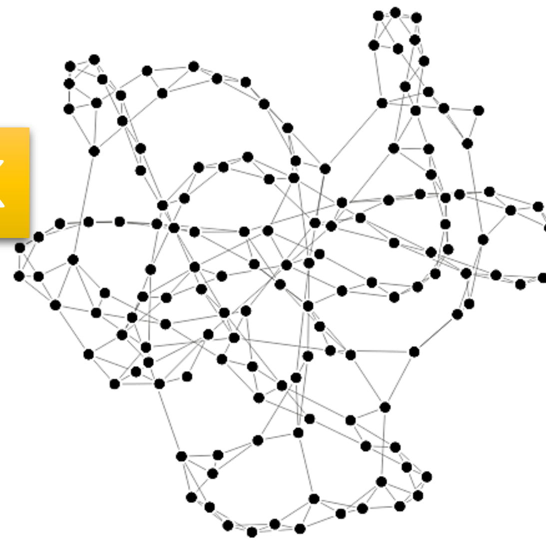
tree-11
tree-17



grid-150
grid-250



sf-100K



gnp-0.001
gnp-0.01
gnp-0.1
gnp-0.5

Real-Life Test Graphs

patent

- US Patent citation network 1975-1999.

wiki

- A subgraph of Wikipedia knowledge graph.

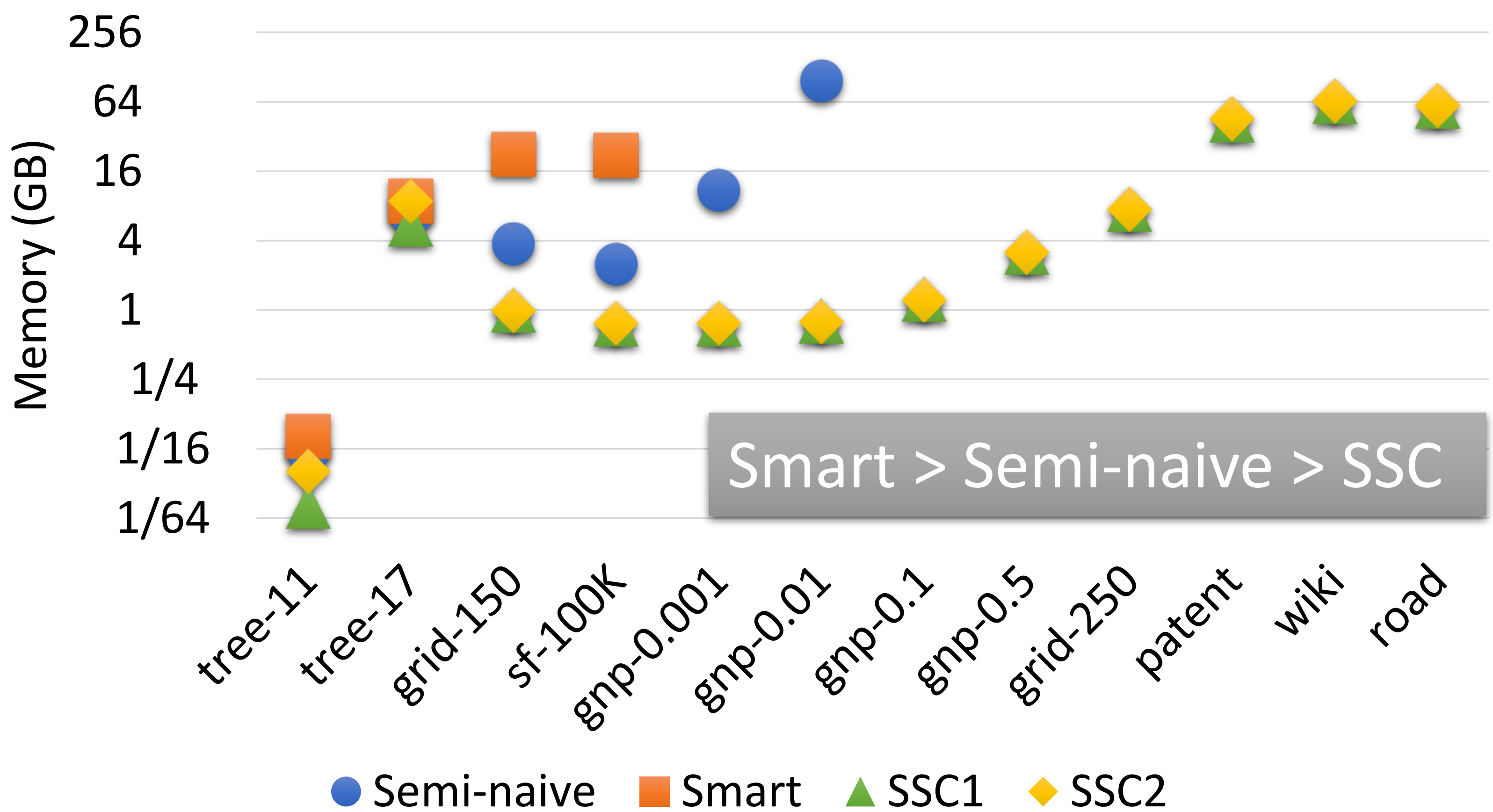
road

- Eastern USA road network.

stanford

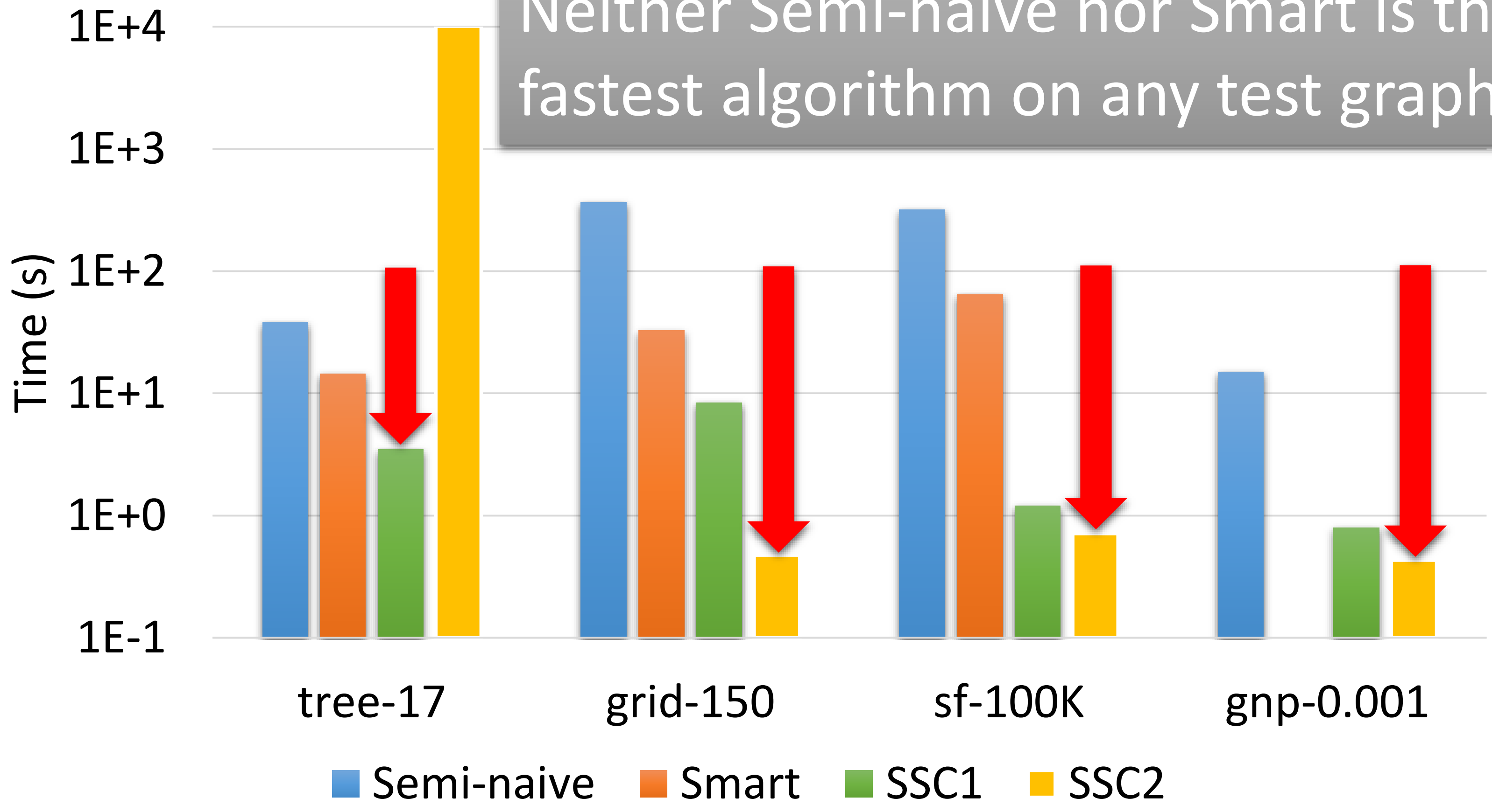
- Stanford web graph from 2002.

Memory Utilization



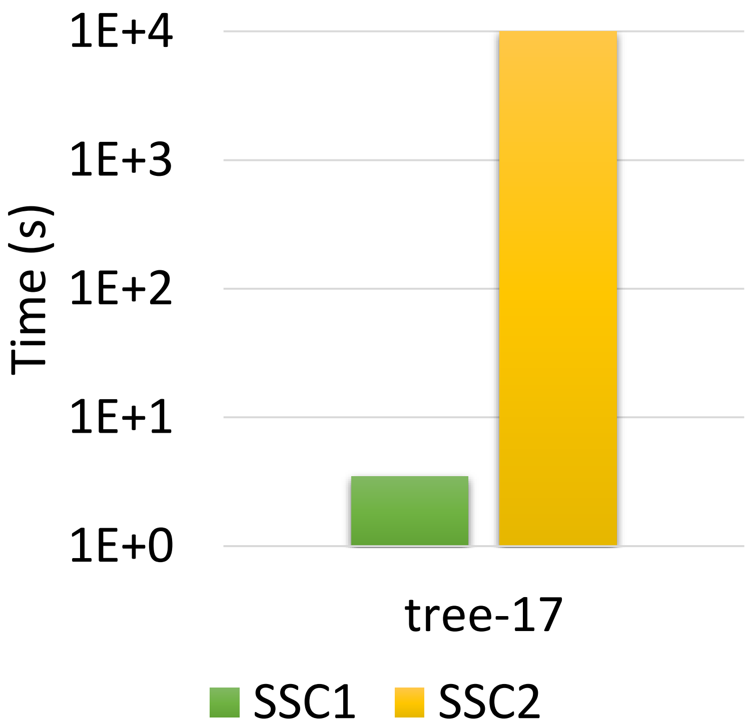
Optimal Execution Time

Neither Semi-naive nor Smart is the fastest algorithm on any test graph.

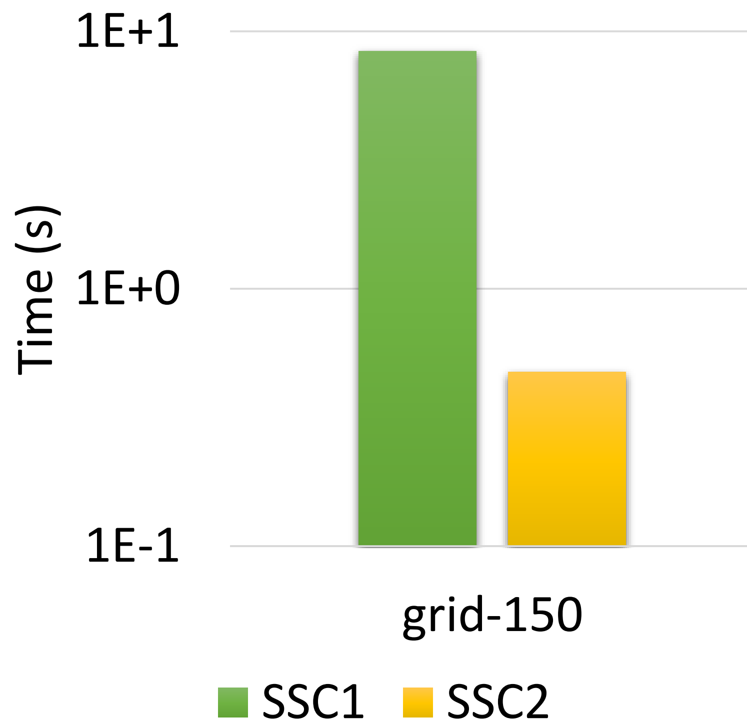


Execution Time Comparison

The array initialization cost dominates the total time.



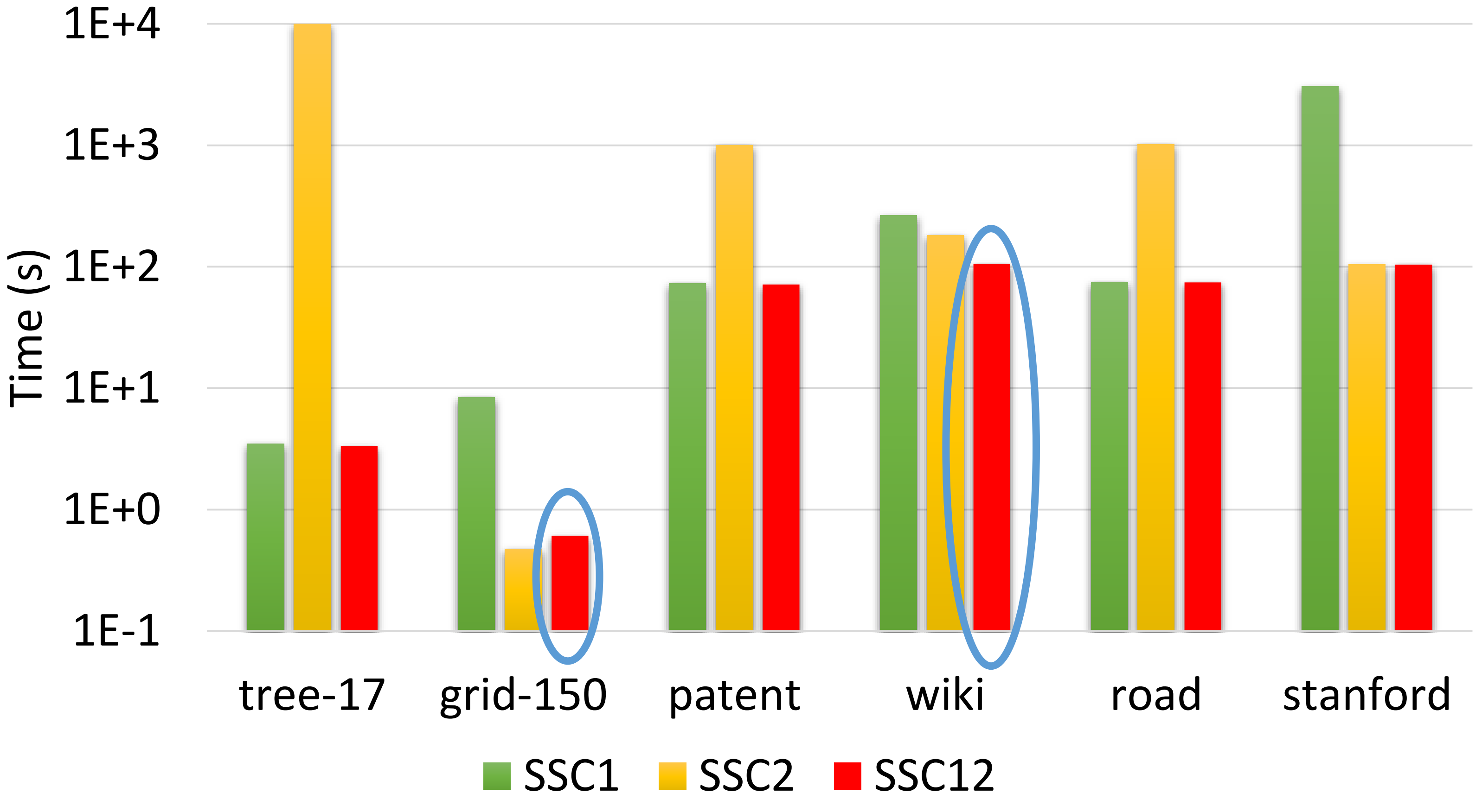
Set representation used by SSC1 becomes less efficient.



Hybrid Algorithm – SSC12

- Combine the advantage of SSC1 and SSC2.
- Start with SSC1, switch to SSC2 if the predicted number of set operations is above a certain threshold.
- Always select the better algorithm (between SSC1 and SSC2) for each vertex in the graph.

Optimal Execution Time



Conclusion & Future Work

- The simple SSC12 algorithm proposed in this paper is an ideal choice for main memory multicore evaluation of recursive query (expressing TC and similar computation).
- Future work
  - Investigate more algorithms, and derive simple criteria for deciding which system can be most cost-effective for the application at hand.
  - Integrate these results into DeALS towards an efficient and scalable system on multicore machines.

Questions