CIFellows 2020-2021

Computing Innovation Fellows

David Tench

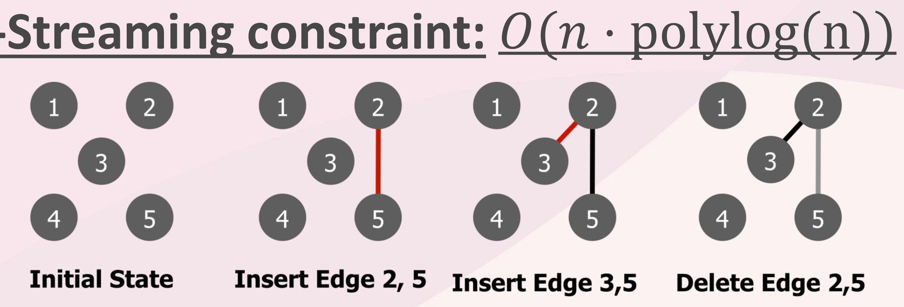
Rutgers University

GraphZeppelin: Processing Enormous, Changing Graphs

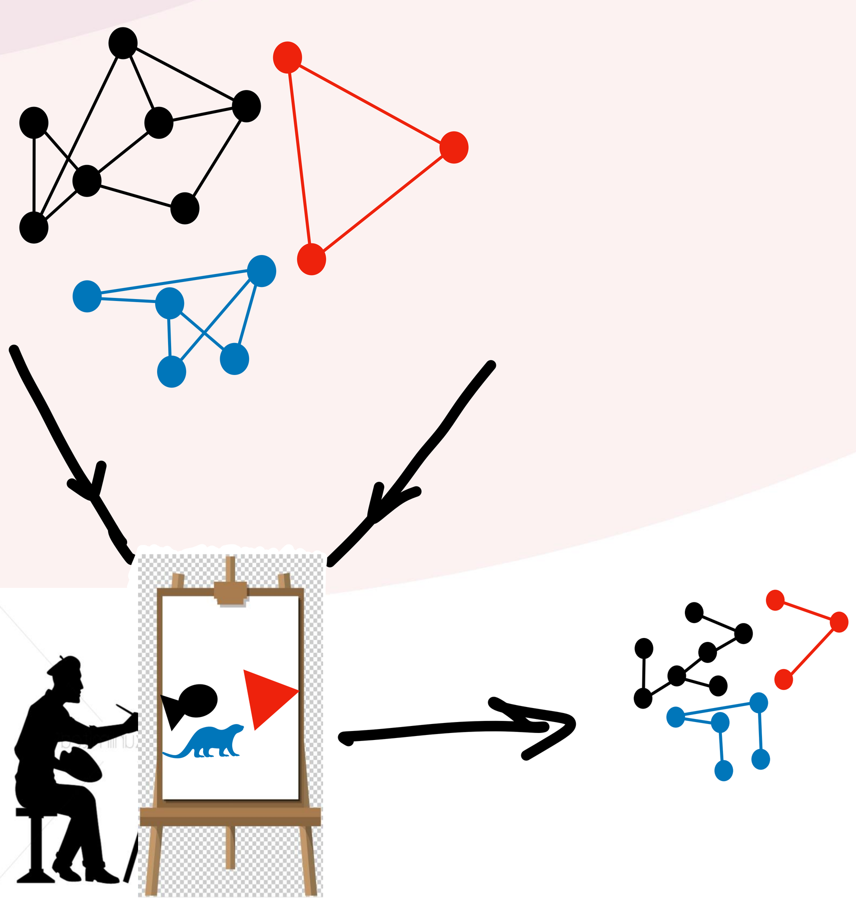
Analyzing Massive Evolving Graphs

Example Problem: find connected components of graph with n nodes subject to ***stream* of edge insertions & deletions.**

Semi-Streaming constraint: $O(n \cdot \text{polylog}(n))$ space.



Sketching solves the problem (in theory)

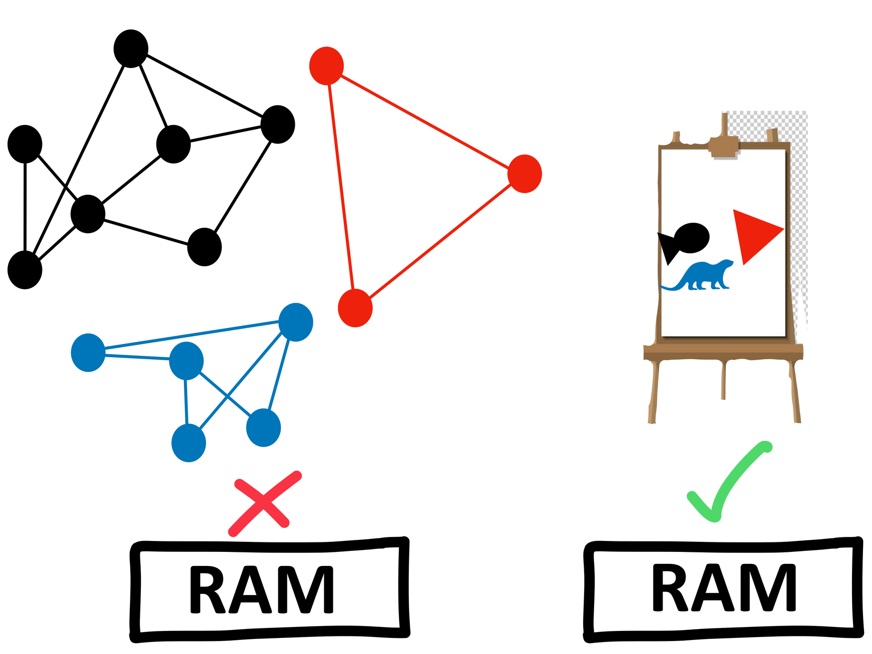


Compressing graph stream via ***linear sketching*** uses $O(n \cdot \log^3(n))$ **space**.

Even though it compresses insert/delete updates one by one in stream order, it can recover connected components **w.h.p.**

[Ahn, Guha, McGregor SODA 2012]

A graph sketching system should:



Handle massive graphs: low space complexity means larger graphs can be processed given fixed RAM size.

Ingest fast streams: low update time crucial for massive graphs that may change millions of times per second.

Solve many graph problems: CC is a black box for many other semi-streaming algs.

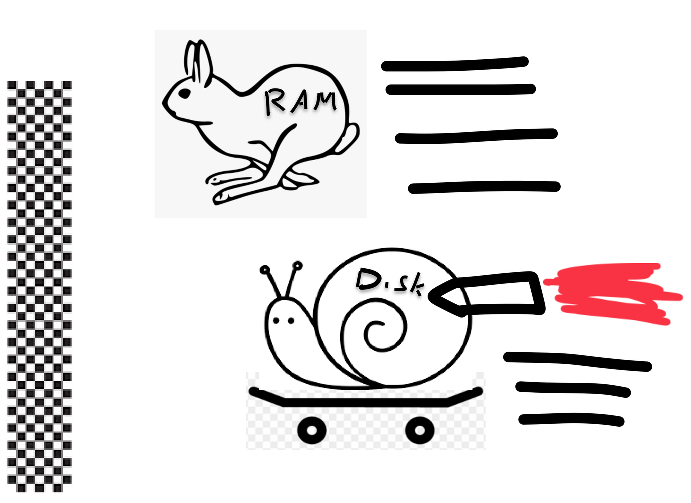
… but existing algorithms don't achieve this.

Sketches are asymptotically small, but how large are they in practice?

Back of envelope calculation for graph on 1 billion nodes:

$$10^9 \cdot \log^3(10^9) = 2.7 \cdot 10^{13}$$

Before constants, requires roughly **25 TB. Too big for RAM!**



Streaming assumption: **only RAM** is fast enough to keep up with high-speed streams.

But **today's high speed SSDs are catching up**: sequential SSD bandwidth approaching random RAM bandwidth.

Can we get sketching to work on disk – without being massively slower?

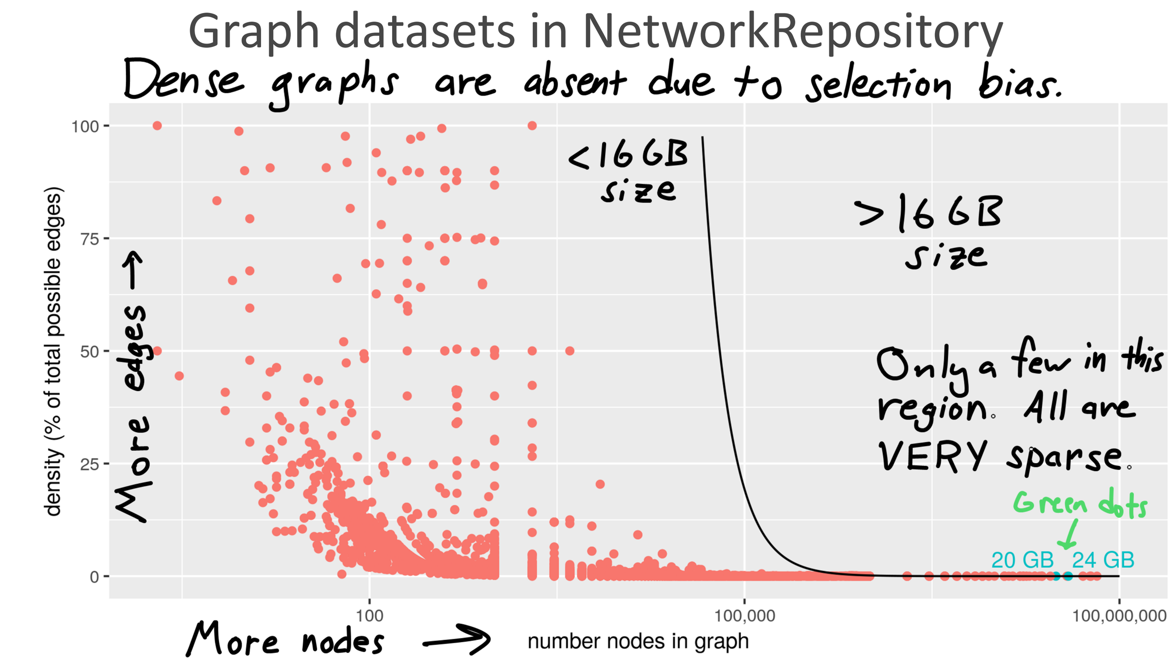
An Aside:

Since sketch size scales with node count but not edge count, they're most useful on **dense** graphs.

Common folk wisdom: only sparse graphs exist at scale.

More likely: dense graphs aren't studied because we lack the tools to work with them.

Sketching makes working with dense graphs possible.



A New Sketching Model

Semi-streaming model*:

$O(\text{polylog}(n))$ RAM

$O(n \cdot \text{polylog}(n))$ fast disk with block size B.

These disks are **larger than RAM**, but can't be made as enormous as old-fashioned hard drives. So we **can't fit an entire dense graph** on them.

As in the external memory model, data on disk is partitioned into ***blocks*** of size B. Data can only be read/written **a block at a time**.

A block I/O costs as much as $O(B)$ RAM accesses.

In addition to **small space** and few passes, we also now want our algorithm to be **I/O efficient**.

A Disk-Friendly Graph Sketching System

Graf-Zeppelin
NOT the Hindenberg
Did NOT explode

GraphZeppelin: a C++ system that solves the connected components problem on graph streams. ("avoiding the data explosion in graph streams")

Its core algorithm is a sketching algorithm that is also I/O-optimal in the external memory model, so it is fast even when run on modern SSDs.

Optimized for **dense graphs.**

Fast: 3-5 million updates/sec in RAM and > 2.5 on disk.

Compact: uses 45GB of space to process a >200GB stream of updates for a 2^{18}-node graph.

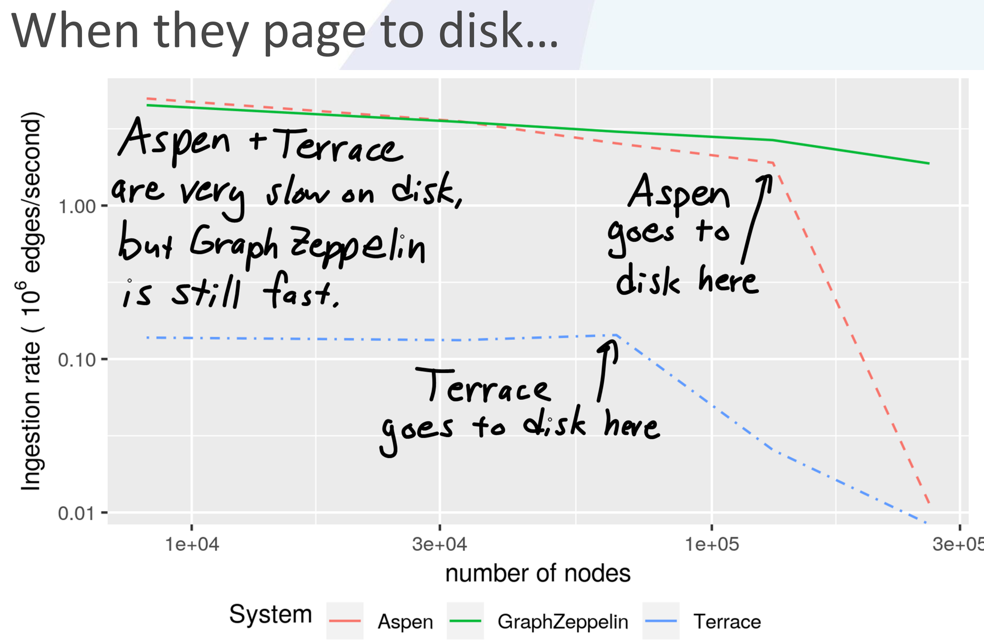
Existing graph stream systems are **optimized for sparse graphs** and store the graph **explicitly**.

Aspen [DBS 2019] **Terrace [PWXB 2021]**

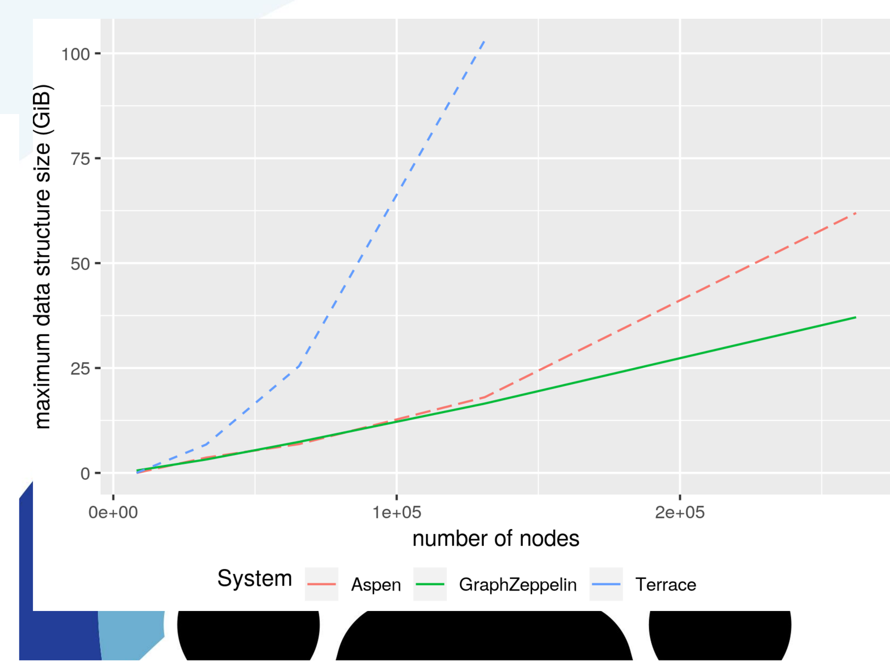
Faster: Aspen and Terrace are very fast on sparse graphs (10-50x10^6 edges/sec) in RAM.

GZ 2x faster than Aspen and 30x faster than Terrace on dense graphs in RAM.

When they page to disk…



More compact: GraphZeppelin uses **half the space of Aspen** and **one tenth the space of Terrace** even for moderately sized dense graphs.



We need to rethink the graph streaming model

$O(n \cdot \text{polylog}(n))$ space is too large for modern RAM. And disk is fast enough to keep up with high-speed streams, if algorithms are I/O efficient.

If you want a streaming/sketching algorithm to be practical, it should also be designed as an external memory algorithm.

CRA Computing Research Association
CCC Computing Community Consortium Catalyst
NSF