Graphs in Machine Learning

Large-Scale Machine Learning on Graphs

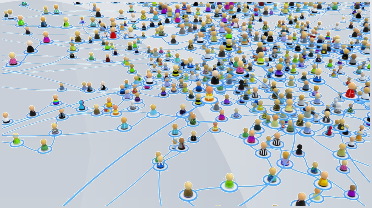
Computational Bottlenecks and Challenges

Michal Valko

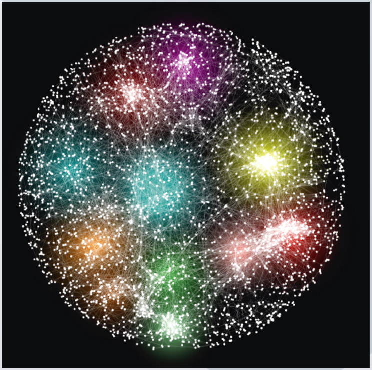
Inria & ENS Paris-Saclay, MVA

Partially based on material by: Rob Fergus, Nikhil Srivastava, Yiannis Koutis, Joshua Batson, Daniel Spielman

Large scale Machine Learning on Graphs

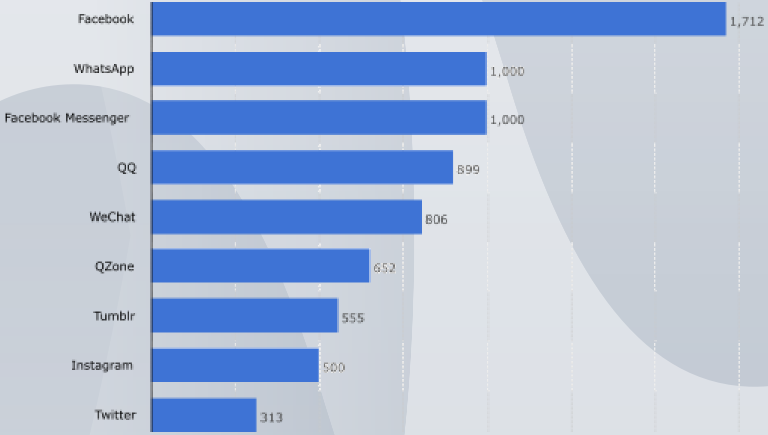


http://blog.carsten-eickhoff.com



Botstein et al.

Are we large yet?



"One **trillion** edges: graph processing at Facebook-scale."
Ching et al., VLDB 2015

Computational bottlenecks

In theory:

Space

$[\mathcal{O}(m), \mathcal{O}(n^2)]$ to store

Time

$\mathcal{O}(n^2)$ to construct

$\mathcal{O}(n^3)$ to run algorithms

Computational bottlenecks

In theory:

Space

$[\mathcal{O}(m), \mathcal{O}(n^2)]$ to store

Time

$\mathcal{O}(n^2)$ to construct

$\mathcal{O}(n^3)$ to run algorithms

In practice:

- 2012 Common Crawl Corpus:

  3.5 Billion pages (45 GB)

  128 Billion edges (331 GB)

Computational bottlenecks

In theory:

Space

$[\mathcal{O}(m), \mathcal{O}(n^2)]$ to store

Time

$\mathcal{O}(n^2)$ to construct
$\mathcal{O}(n^3)$ to run algorithms

In practice:

- 2012 Common Crawl Corpus:
  3.5 Billion pages (45 GB)
  128 Billion edges (331 GB)
- Pagerank on Facebook Graph:
  3 minutes per iteration, hundreds of iterations, tens of hours on 200 machines, run once per day

Two phases

1. Preprocessing:

Two phases

1. Preprocessing:

   From vectorial data: Collect a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, construct a graph $\mathbf{G}$ using a similarity function

Two phases

1. Preprocessing:

   From vectorial data: Collect a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, construct a graph $\mathbf{G}$ using a similarity function
   Prepare the graph: Need to check if graph is connected, make it directed/undirected, build Laplacian

Two phases

1. Preprocessing:

   From vectorial data: Collect a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, construct a graph $\mathbf{G}$ using a similarity function

   Prepare the graph: Need to check if graph is connected, make it directed/undirected, build Laplacian

   Load it on the machine: On a single machine if possible, if not find smart way to distribute it

Two phases

1. Preprocessing:
   From vectorial data: Collect a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, construct a graph $\mathbf{G}$ using a similarity function
   Prepare the graph: Need to check if graph is connected, make it directed/undirected, build Laplacian
   Load it on the machine: On a single machine if possible, if not find smart way to distribute it

2. Run your algorithm on the graph

Large scale graph construction

Time bottleneck

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow

Large scale graph construction

Time bottleneck

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow

Large scale graph construction

Time bottleneck

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow
- In both cases bottleneck is the same, given a node finding close nodes (k neighbours or ε neighbourhood)

Large scale graph construction

Time bottleneck

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow
- In both cases bottleneck is the same, given a node finding close nodes (k neighbours or ε neighbourhood)

Large scale graph construction

Time bottleneck

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow
- In both cases bottleneck is the same, given a node finding close nodes (k neighbours or ε neighbourhood)

Fundamental limit: just looking at all similarities already too slow.

Large scale graph construction

Time bottleneck

- Constructing k-nn graph takes $\boldsymbol{O}(n^2 \log(n))$, too slow
- Constructing ε graph takes $\boldsymbol{O}(n^2)$, still too slow
- In both cases bottleneck is the same, given a node finding close nodes (k neighbours or ε neighbourhood)

Fundamental limit: just looking at all similarities already too slow.

Can we find close neighbours without checking all distances?

Large scale graph construction

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

Large scale graph construction

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization

Large scale graph construction

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization
- KD-Trees – Cover Trees – NN search is $\mathcal{O}(\log N)$ per node

Large scale graph construction

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization
- KD-Trees – Cover Trees – NN search is $\mathcal{O}(\log N)$ per node
- Locality Sensitive Hashing (LSH)

Large scale graph construction

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization
- KD-Trees – Cover Trees – NN search is $\mathcal{O}(\log N)$ per node
- Locality Sensitive Hashing (LSH)

Large scale graph construction

Split your data in small subset of close points

Can find efficiently some (not all) of the neighbours.

- Iterative Quantization
- KD-Trees – Cover Trees – NN search is $\mathcal{O}(\log N)$ per node
- Locality Sensitive Hashing (LSH)

More general problem: learning good codeword representation

Storing graph in memory

Main bottleneck: **space**.

As a Fermi (back-of-the-envelope) problem

- Storing a graph with m edges require to store m tuples $(i, j, w_{i,j})$ of 64 bit (8 bytes) doubles or int.

Storing graph in memory

Main bottleneck: **space**.

As a Fermi (back-of-the-envelope) problem

- Storing a graph with m edges require to store m tuples $(i, j, w_{i,j})$ of 64 bit (8 bytes) doubles or int.
- The largest compute-optimized cloud instances have 64$+$ cores, but only 128 GB of memory.

Storing graph in memory

Main bottleneck: **space**.

As a Fermi (back-of-the-envelope) problem

- Storing a graph with m edges require to store m tuples $(i, j, w_{i,j})$ of 64 bit (8 bytes) doubles or int.
- The largest compute-optimized cloud instances have 64$+$ cores, but only 128 GB of memory.
- We can store $128 * 1024^3/(3 * 8) \sim 5.7 \times 10^9$ (5.7 billion) edges in a single machine memory.

Storing graph in memory

But wait a minute

- Natural graphs are sparse.

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
  Subcomponents are very dense, and they grow denser over time

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  - ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
    Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
  Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  ↳ Losing large scale relationship, losing regularization

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  - ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
    Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  - ↳ Losing large scale relationship, losing regularization
- I will split my graph across multiple machines

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
  Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  ↳ Losing large scale relationship, losing regularization
- I will split my graph across multiple machines
  ↳ Your algorithm does not know that.

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
  Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  ↳ Losing large scale relationship, losing regularization
- I will split my graph across multiple machines
  ↳ Your algorithm does not know that.
  What if it needs nonlocal data? Iterative algorithms?

Storing graph in memory

But wait a minute

- Natural graphs are sparse.
  ↳ For some it is true, for some it is false (e.g. Facebook average user has 300 friends, Twitter averages 208 followers)
  Subcomponents are very dense, and they grow denser over time
- I will construct my graph sparse
  ↳ Losing large scale relationship, losing regularization
- I will split my graph across multiple machines
  ↳ Your algorithm does not know that.
  What if it needs nonlocal data? Iterative algorithms?
  More on this later

Michal Valko

michal.valko@inria.fr

Inria & ENS Paris-Saclay, MVA

https://misovalko.github.io/mva-ml-graphs.html