On Parameterized Vertex Cover in Streaming

Adam Cox

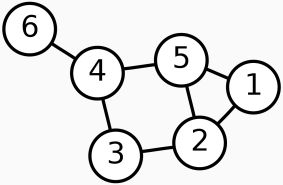
July 6, 2020

School of Computer Science
University of Birmingham

Graph Theory

Nodes/Vertices
Representing objects

Edges
Representing relationships between objects

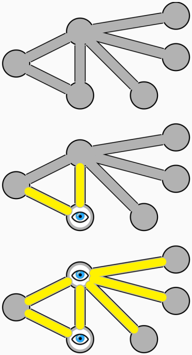


Together this makes a **graph**

Vertex Cover

A set of vertices that includes at least one endpoint of every edge of the graph.

A real life example would be putting up cameras at road intersections to be able to see down every road.



Karp's 21 NP-complete problems

The vertex cover problem is an NP-complete problem: it was one of Karp's 21 NP-complete problems.

It is often used in computational complexity theory as a starting point for NP-hardness proofs.

Parameterized Algorithms

Classical complexity measures runtime of an algorithm solely based on the input size.

Parameterized complexity measures runtime of an algorithm based on parameters of the input or output.

Non-parameterized/classical time complexity has been used since the 1960s. Parameterized time complexity only came about in the 1990s.

Parameterized Vertex Cover

This is known as the **Vertex Cover Problem** or k-VC

k-VC
INSTANCE: Graph G and positive integer k
QUESTION: Does G have a vertex cover of size at most k?

The vertex cover problem is an NP-complete problem

Big Data and Streaming

Big Data applies to any dataset that is too big to store/look at in one go. We need other methods to be able to make the data usable.

Non-parameterized space complexity began being studied around the 2000s and only in 2014 did parameterized space complexity start being covered.

Parameterized Vertex Cover in Streaming

So for the full problem, we are looking at:

- Insertion-only stream of edges
- Value k

And we want a True/False answer. In practise, users would also want the set of vertices that make up the vertex cover.

Historically

Branching

- Easy logarithmic complexity
- Simple implementation

Kernelization

- Simple rules
- Powerful if done right

State of the Art

In 2014, Chitnis et al put forward streaming algorithms based on these two ideas.

Branching - Space required is $O(k \cdot \log n)$ and requires 2^k passes

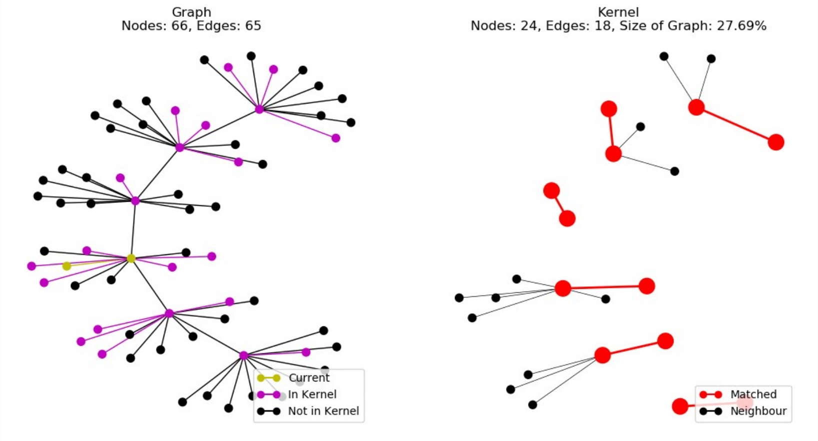
Kernel - Space required is $O(k^2)$ and requires 1 pass

Implementation

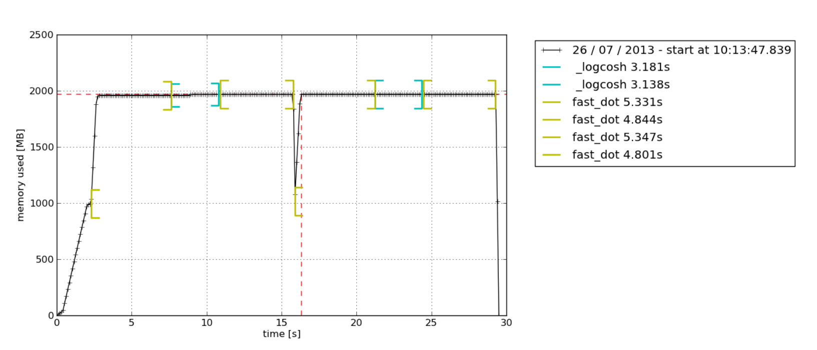
So I built these streaming algorithms into a number of domains:

- Local
- Local-Stream
- Stream

Local - Visualisation

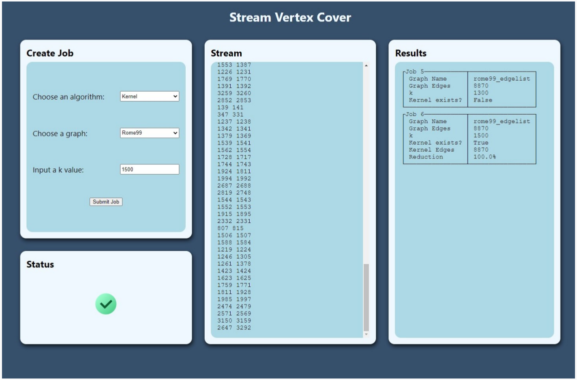


Local-Stream - Performance Profiling



Disclaimer: This is just an example of what a memory profiling graph looks like.

Stream - Graph Streaming Platform



Stream Vertex Cover

Create Job

Choose an algorithm: Kernel

Choose a graph: Rome99

Input a k value: 1500

Status

✓

Stream

```
1553 1387
1226 1231
1769 1770
1391 1392
3259 3260
2852 2853
139 141
347 331
1237 1238
1342 1341
1379 1369
1539 1541
1562 1554
1728 1717
1744 1743
1924 1811
1994 1992
2687 2688
2819 2748
1544 1543
1552 1553
1915 1895
2332 2331
807 815
1506 1507
1588 1584
1219 1224
1246 1305
1261 1378
1423 1424
1623 1625
1759 1771
1811 1928
1985 1997
2474 2479
2571 2569
3150 3159
2647 3292
```

Results

Job 5

Graph Name	rome99_edgelist
Graph Edges	8870
k	1300
Kernel exists?	False

Job 6

Graph Name	rome99_edgelist
Graph Edges	8870
k	1500
Kernel exists?	True
Kernel Edges	8870
Reduction	100.0%

kafka

Thank you - Any questions?