Sheep: A Scalable Distributed Graph Partitioner

Daniel Margo *dmargo@eecs.harvard.edu* Margo Seltzer *margo@eecs.harvard.edu*

Problem

Graph partitioning enables efficient computing on very large graphs. But how can we efficiently find partitions of graphs in the absence of an *a priori* partitioning?

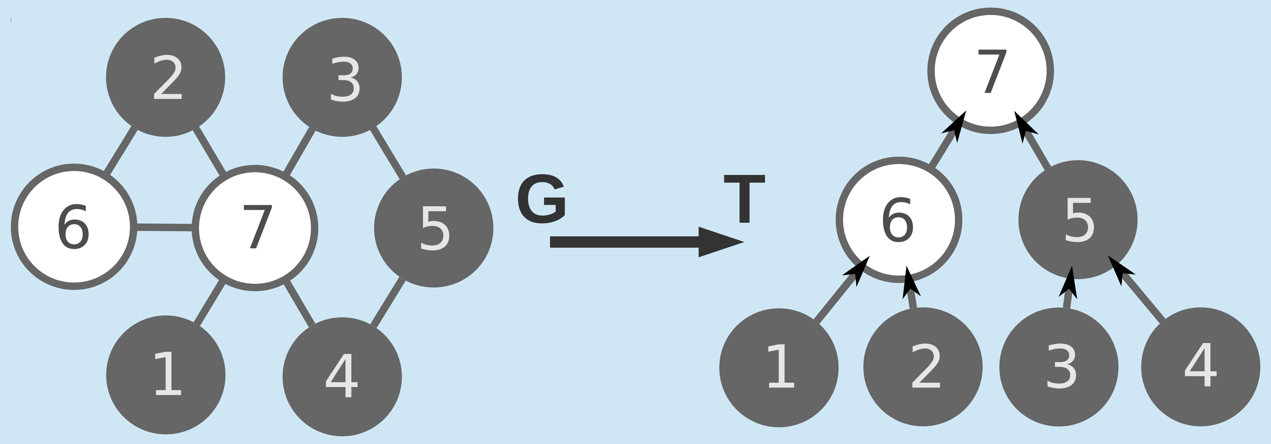
Solution

Sheep is a scalable distributed graph partitioning algorithm with a map-reduce structure. Sheep's runtime and results are independent of *a priori* partitions, so the input graph can be arbitrarily distributed among jobs.

Overview

Sheep partitions a graph by:

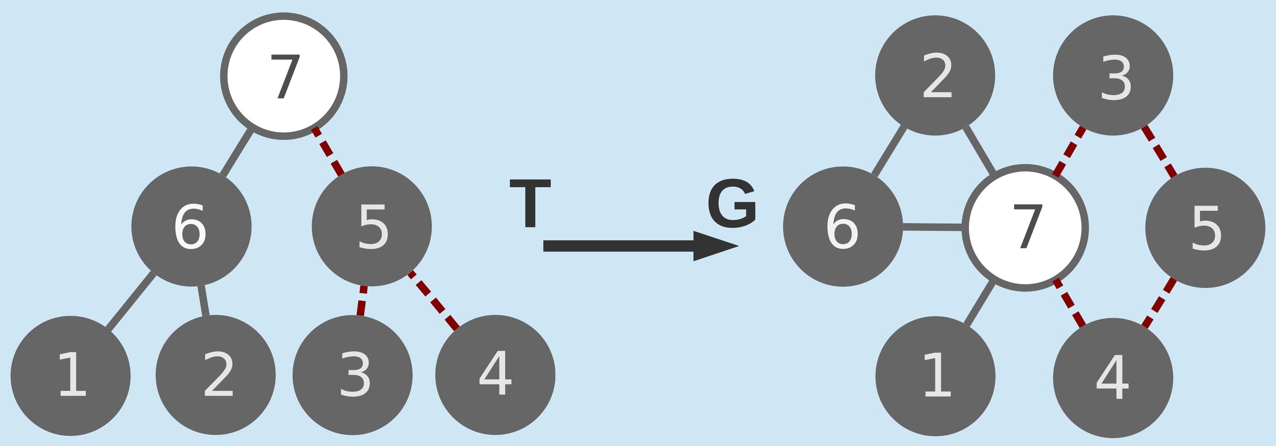
1. Sorting the vertices,
2. Reducing the graph to an *elimination tree*,
3. Partitioning that elimination tree, and
4. Translating the result into graph partitions.



An elimination tree T of a graph G is a rooted tree where: *if (X,Y) in G, then X is below Y in T or vice-versa.*

Partitioning

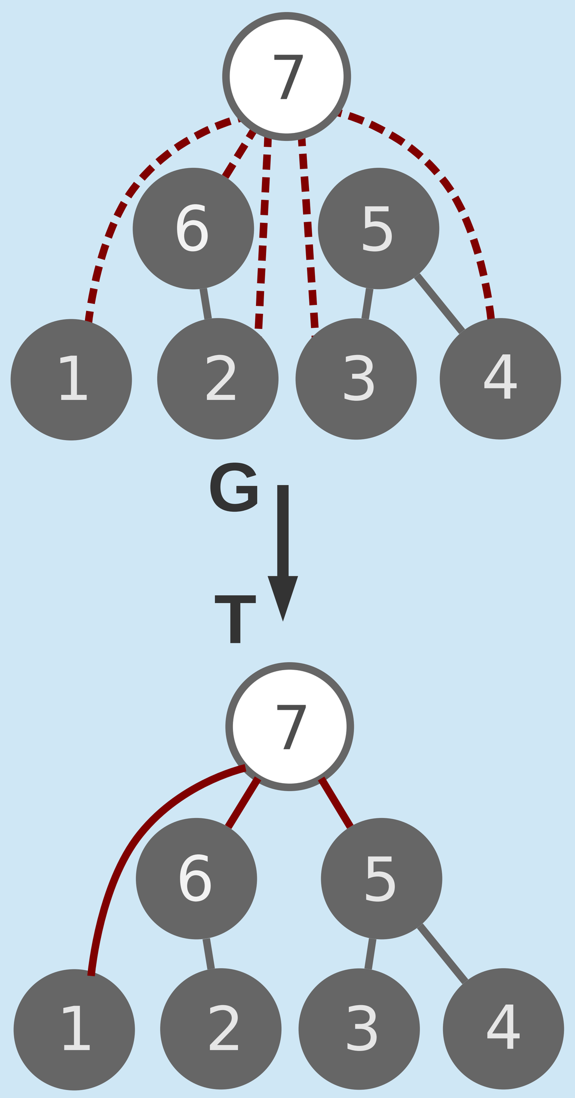
Sheep is an *edge partitioner* that minimizes *communication volume.* CV measures the number of partitions that each vertex communicates with. Vertices that communicate with more than one partition are called *border vertices.* When Sheep partitions the tree, it upper bounds the CV by bounding the set of border vertices.



Example: Vertex 7 is the only possible border vertex between the left and right partition sets.

Algorithm

Sheep creates its tree via a union-find algorithm. The tree depends on the graph and the *vertex sort order*.



Example

Vertex 7 has 5 edges into 3 preceding union-find sets. It adopts one child for each set. Then, X is below 7 for all edges (X,7).

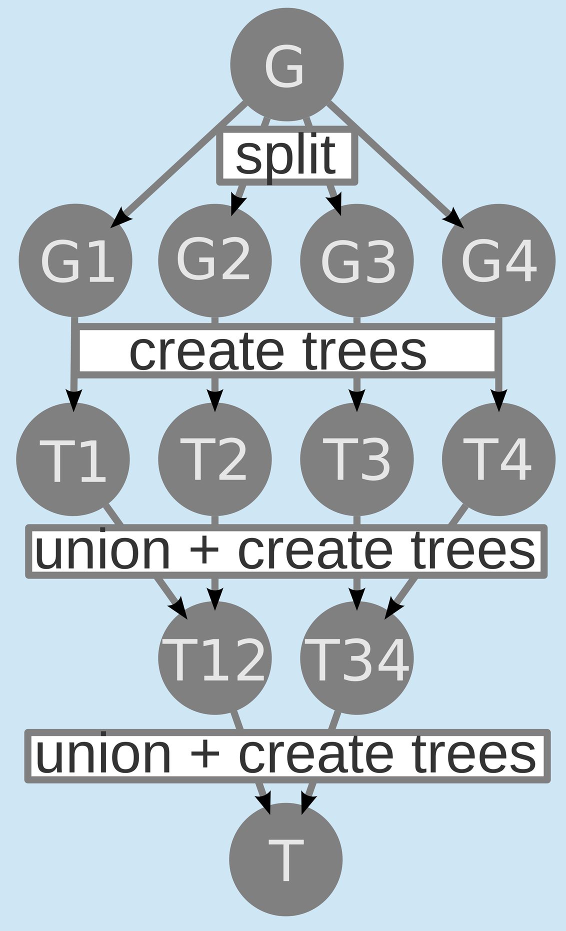
Pseudocode

Let U = (V,P) be a union-find on a set V that chooses as each subset representative the maximum element in that subset by a total order P = (V,<). Let T = (V,T_E) be an elimination tree on a vertex set V.

```
Require: G is an undirected graph (V,E)
Require: P is a total order (V,<)
Function: Persistent_Union_Find(G,P):
   U := (V,P)
   T := (V,null)
   For all z in V in order P do:
      For all (x,z) in E, x < z do:
         y = U.find(x)
         If y != z then:
            U.union(y,z)
            T_E := T_E U (y,z)
```

Distribution

Sheep can split a graph into subgraphs, map trees for each, and then reduce the trees into a final, valid tree for the original graph.

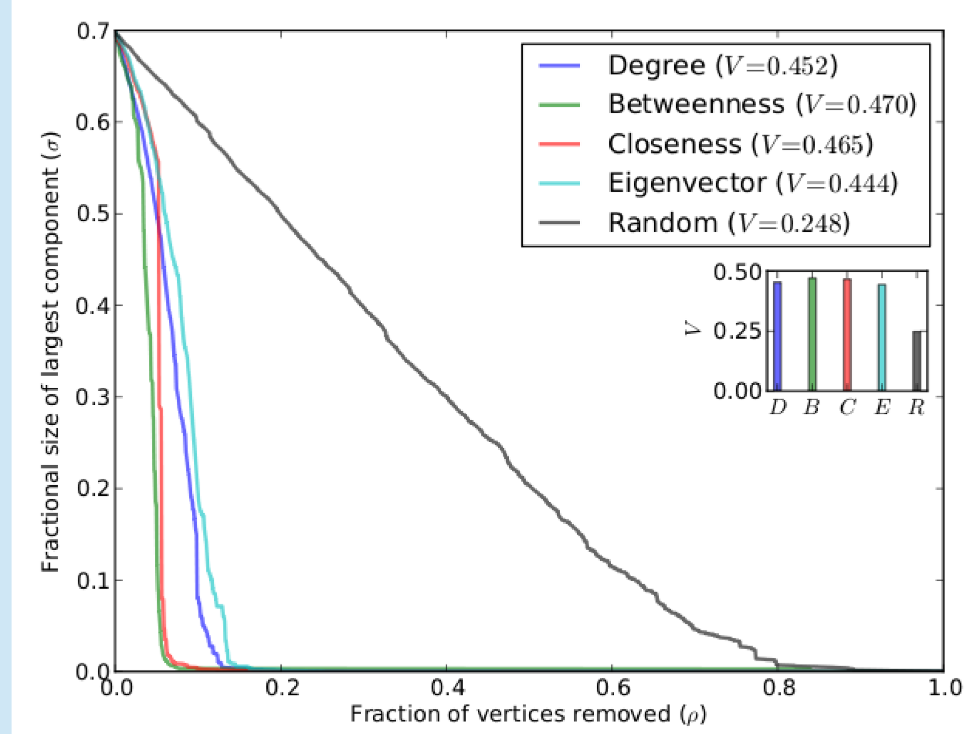


Let T(G,P) be the tree produced by Sheep in order P. Let G1 and G2 be two subgraphs of G such that G1 U G2 = G. Then, we prove that: T(G,P) = T(T(G1,P) U T(G2,P), P)

Sorting

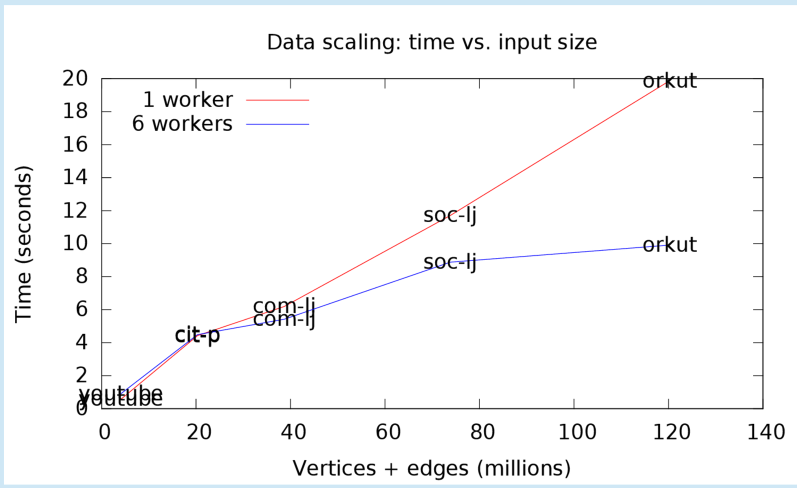
Sheep's partitions are of highest quality when the elimination tree is short.

It is well-known that "power law" graphs disconnect quickly if one deletes vertices in degree order **[1]**. We show that this is equivalent to finding a short e-tree.

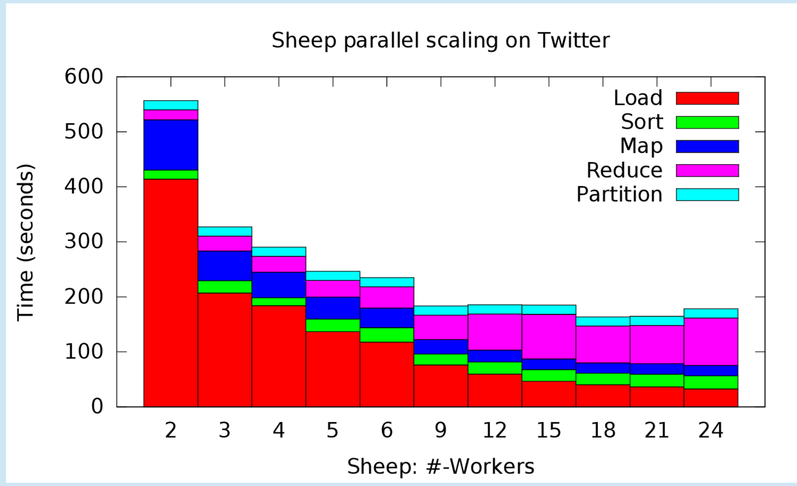


Decay orders on a Physicscollaboration network **[1]**.

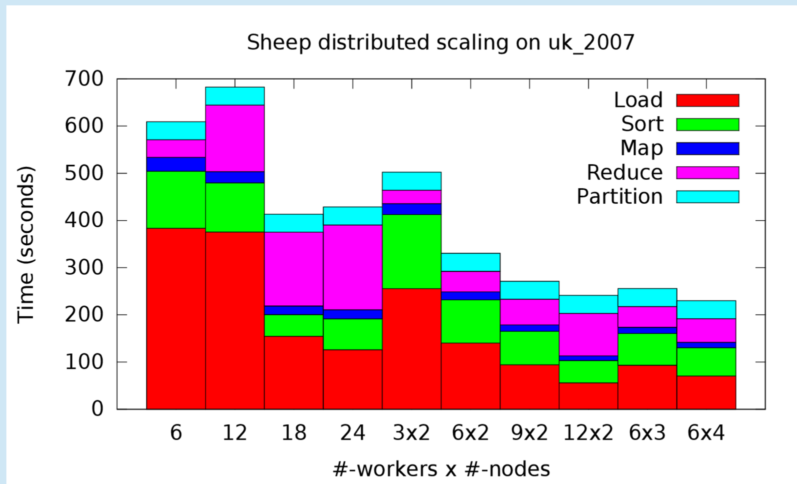
Sheep is fast and scalable!



8GB commodity machine with SSD
Sheep uses cores efficiently on in-memory graphs. *Sheep is up to 6 times faster than METIS on these graphs*. Sheep can process out of memory by splitting the graph into working sets.

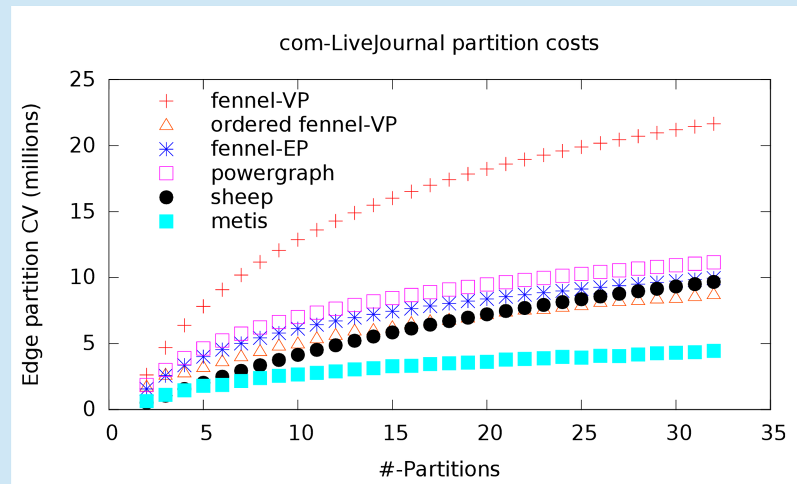


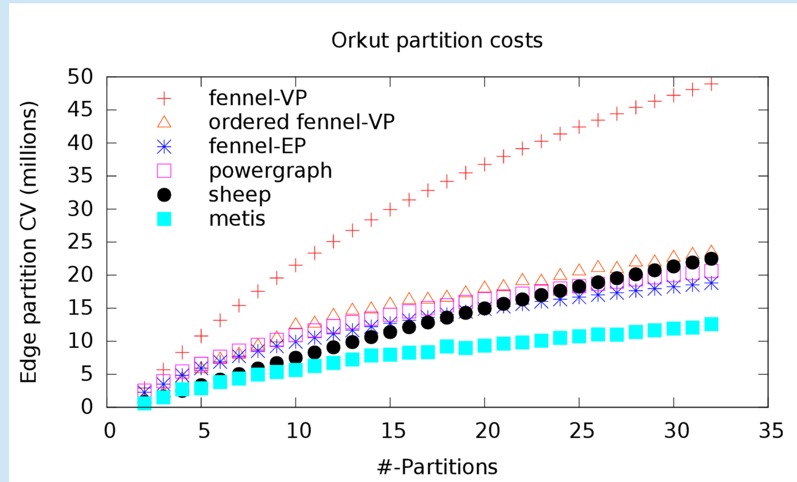
256GB cluster node with Infiniband
42 million vertices, 1.5 billion edges
Scaling is limited by the reduce step. *Sheep is more than 8 times faster than Fennel,* which takes 22 minutes. METIS cannot process this graph in 256GB.

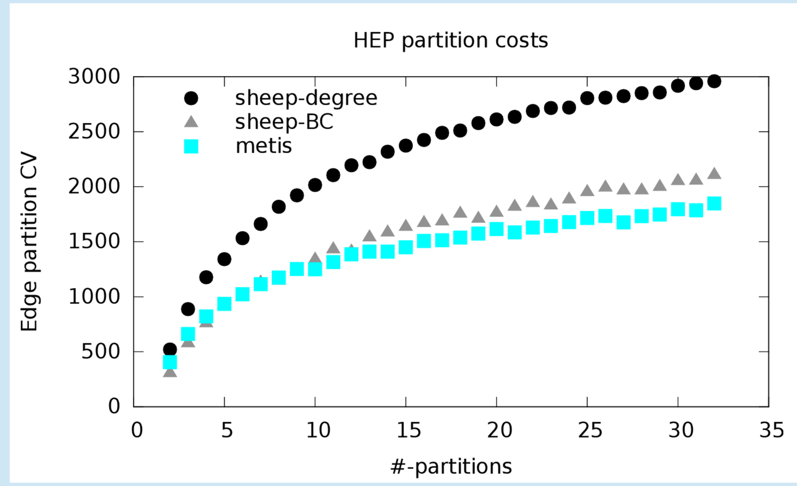


106 million vertices, 3.74 billion edges
Sheep is ultimately data-bound on one node, but it profits further by scaling horizontally across memory controllers.

Sheep finds good partitions!







Sheep's partition costs are competitive with other partitioners. METIS produces better partitions, but at great expense. Sheep is even competitive with METIS for small partition counts. *Quality data is in part reproduced from work by Bourse et al.* **[2]**.

References

D. Margo, M. Seltzer. A scalable distributed graph partitioner. *Proceedings of the VLDB Endowment*, Vol. 8, No. 12.

[1] S. Iyer, T. Killingback, B. Sundaram, and Z. Wang. Attack robustness and centrality of complex networks. *PloS ONE*, 8(4):e59613, 2013.

[2] F. Bourse, M. Lelarge, and M. Vojnovic. Balanced graph edge partition. *20th ACM International Conference on Knowledge Discovery and Data Mining*, p. 1456-1465. ACM, 2014.

Sheep's partitions improve with a better sort order. With a high-quality order, Sheep's partitions are comparable to METIS. At present these orders are expensive to compute, but this is exciting for future work in e.g. graph database cracking.

For more info, please see our paper in VLDB'15!
https://github.com/dmargo/sheep