# Maintaining Connectivity in Parallel Graph Partitioning

Christopher I. Jones
jonesc10@rpi.edu

Ian Bogle
boglei@rpi.edu

George M. Slota
slotag@rpi.edu

Rensselaer

## Motivation

**Partitioning** graphs is an important task in parallel scientific computing and graph analysis applications to balance work per-task and minimize communication. For large graphs and meshes, parallelizing the partitioning process is heavily desired. If parts within a partition remain **connected**, they can allow simpler computations and reduce the number of very small components that are disconnected from the rest of their part.

Applications for partitioning:

- Parallel graph analysis using **clusters**
- **Scientific computing** of fine meshes, which desires connectivity and multi-weight vertices
- Problem solving in biology, power grids, image processing, and many more [3]
- **Redistricting**, which requires connectivity, and can reduce *gerrymandering* issues [2]

## Background

Graph partitioning typically is the process of creating vertex-disjoint sets on a graph that minimize edge cut and constrain weight per-part. This project focuses on adding an additional constraint: **connectivity**. It is common for partitioning algorithms to not guarantee connected partitions, but it can be a highly desired feature.

Several categories of graph partitioning algorithms exist. **Global** algorithms work with the entire graph and compute a direct solution, and they're mainly used for smaller graphs. **Iterative improvement** heuristics gradually improve partitions from an initial starting position. These take a small set of vertices at a time and move them to the best suited part. **Multilevel** algorithms condense the graph small enough to do an initial partitioning, then work back and uncondense the graph [3].

The work done in this project is built on an existing algorithm: **Partitioning using Label Propagation** (**PuLP**) [1]. PuLP has achieved good vertex balancing and edge cut, while obtaining greatly improved runtime and memory usage over other partitioning algorithms. The results of this will be integrated into PuLP to address connectivity issues.

## Methods for Maintaining Connectivity

We are building upon an existing PuLP [1] algorithm, adding features and functionality to aid connectivity. The original algorithm has 2 inner iterations:

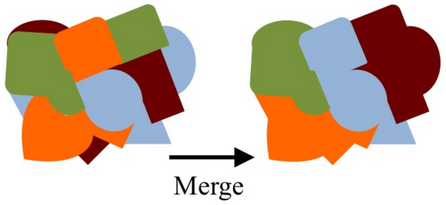
- **Balance** focuses on balancing the vertices between parts. Each vertex considers the *gain* to being in part $P$:

$$gain(P) = n_P \cdot w_P$$

  $n_P$ is the number of neighbors in $P$. $w_P$ is how underweight $P$ is.

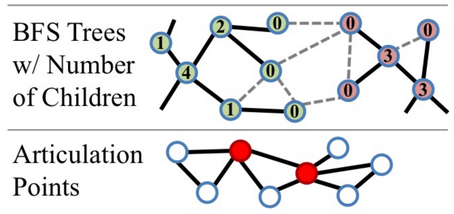
- **Refine** works on minimizing the edge cut relative to each vertex.

### Merge Small Components




Ideally, each part has one component. Every some number of iterations, merge all components that are not the largest in their part into a neighboring component.

### Restrict Vertex Movement




Calculate BFS trees on each component, and only allow the **leaves** to move. Or, every so many vertex movements, calculate bi-connectivity, and prevent **articulation points** from moving.

### Additional Metrics

Add a new metric to the *gain* equation by 2 methods:

(1) $gain(P) = (n_P + \boldsymbol{a_P}) \cdot w_P$

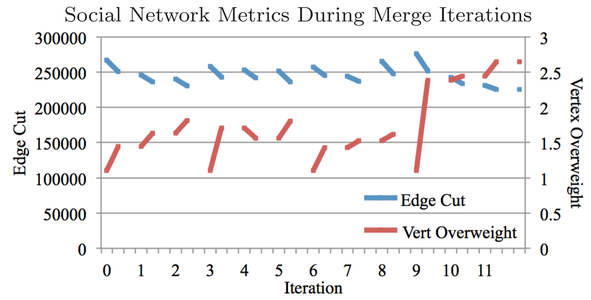
(2) $gain(P) = n_P \cdot w_P / \boldsymbol{a_P}$

$$a_P = \sum_{\substack{u \in N(v) \\ u \in P}} \frac{children(u)}{d(u)}$$

Where $N(v)$ is the neighborhood of $v$, $children(u)$ is the number of children $u$ has in the component's BFS tree, and $d(u)$ is the degree of $u$.

## Results

### Merge Small Components




Merging works decently for some graphs, such as a road network, while sacrificing some of the vertex balancing and edge cut. But for other complex graphs, like a social network, vertices are thrown way off balance.

### Restrict Vertex Movement

| $C$ | Vertex Overweight | Edge Cut | Small Comp. Count |
|---|---|---|---|
| 0 | 1.5-2.5 | >1,000K | 0 |
| 1 | 1.1-1.2 | 800K | 4,000 |
| 2 | $\leq$ 1.1 | 650K | 5,000 |
| $\infty$ | <1.1 | 150K | 5,500 |

**Leaves Only Method:** Results are from a web-crawl graph where only vertices with $C$ or less children are allowed to move.

| $R$ | 10K | 2.5K | 1,250 | 320 | 80 |
|---|---|---|---|---|---|
| Comp. Count | 2,000 | 1,500 | 1,400 | 1,275 | 1,200 |

**Articulation Point Method:** Running on a road network graph, biconnectivity is recalculated every $R$ vertex updates. The original algorithm outputs only around 130 components.

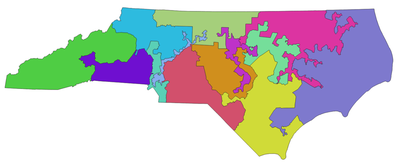
### Additional Metrics

| | Vertex Overweight | Edge Cut | Small Components Count | Small Components Size |
|---|---|---|---|---|
| Normal alg. | 1.039 | 125.4K | 6,100 | 22.17 |
| Method (1) | 1.040 | 125.5K | 5,800 | 22.53 |
| Method (2) | 1.057 | 434.4K | 14,400 | 13.94 |
| **Add Merging as a Final Step:** | | | | |
| Normal alg. | 1.182 | 94.5K | 0 | N/A |
| Method (1) | 1.174 | 95.3K | 0 | N/A |
| Method (2) | 1.081 | 356K | 0 | N/A |

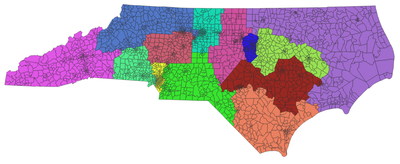
**Using an Additional Metric:** These numbers are from a web-crawl graph when adding the Average Number of Children metric, $a_P$. Method (2) produces many components that are relatively small, allowing those components to be easily merged.

## Application: Political Redistricting

**Gerrymandering** is the issue of politicians manipulating districts based on where voters live. Translating the redistricting problem to graph partitioning, **census blocks** are vertices, **districts** are parts, and edges represent shared borders. Previous work includes PEAR [2], which is an evolutionary algorithm. We've applied PuLP to counties in North Carolina, using population demographics as vertex weights and border lengths as edge weights.



**Gerrymandering:** North Carolina districts from 2013-2016.



**PuLP:** A redistricting using PuLP with small component merging.

Redistricting introduces new metrics that can be taken into consideration, such as compactness and competitiveness, as defined in [2]. **Compactness** is related to how well connected a district is and is defined as the area divided by perimeter squared. **Competitiveness** takes the numbers of voters in opposing parties to measure how evenly they're distributed in the districts. Other demographic data can also be considered.

## Acknowledgements

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute under Contract No. DE-AC02-05CH11231 at Rensselaer Polytechnic Institute and Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

## References

[1] G. Slota, K. Madduri, S. Rajamanickam PULP: Scalable Multi-Objective Multi-Constraint Partitioning for Small-World Networks In *Proc. IEEE BigData Conf.*, 2014.

[2] Y. Y. Liu, W. K. T. Cho, S. Wang PEAR: a massively parallel evolutionary computation approach for political redistricting optimization and analysis. In *Swarm and Evolutionary Computation, 30, 78-92*, 2016.

[3] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, C. Schulz Recent Advances in Graph Partitioning In *Algorithm Engineering*, 2016.