Programmatically Determining Bifurcation Points in Transportation Networks

Kate M. Sanders
Department of Mathematics and Computer Science, Hendrix College

Introduction

Constructing networks to transport goods from production to consumption is a problem with ramifications in fields from economics to ecology. If building a road had no cost, the most efficient transportation network would directly connect the production and consumption sites. However, once the physical costs of a road are considered, it is often advantageous to "carpool" goods for part of the way between the source and sink nodes due to efficiencies of scale. There are many examples of this in nature, such as rivers and the cardiovascular system. It is efficient to carry goods together, then diverge into smaller paths as the destination approaches.

Background

- For any cost-weight, $\alpha \in [0, 1]$, and for any transportation network, T, we may define the transportation cost of T to be its M^α mass:

$$M^\alpha(T) := \sum_{e \in E(T)} [\lambda(e)]^\alpha \mathcal{L}(e).$$

  Where $\mathcal{L}(e)$ denotes the length of edge e. [3]

- Let T and S be two transportation networks with the same source and sink nodes but possibly with different interior bifurcation points. Drawing both T and S encloses a set of regions $\mathcal{R}$. The area of a region $r \in \mathcal{R}$ is $\triangle(r)$. The weight of a region, $\lambda(r)$, is determined by the outer edge with the least weight. Then given an α,

$$Fill^\alpha(T - S) = \sum_{r \in \mathcal{R}} [\lambda(r)]^\alpha [\triangle(r)]$$

- A flow is a method used to find an optimal network by iteratively minimizing a cost function. To create a flow, begin with a fixed step size $h > 0$ and set $\alpha > 0$. In the initial transportation network, T_0, the sources are connected to sinks by straight directed edges. Downes [3] inductively defined a sequence of real-valued transportation paths $\{T_j^h\}$ such that T_j^h minimizes the functional:

$$G_j(T_j^h) = [M^\alpha(T_j^h)]^2 + \frac{[Fill^\alpha(T_{j-1}^h - T_j^h)]^2}{h}.$$

Problem

Using a cost-reducing flow could be useful in transportation network problems such as constructing irrigation systems. However, the process of creating a two-dimensional cost-reducing flow has not yet been automated. To address this need, we have created:

- A user-friendly, interactive online program for evolving and visualizing cost-reducing flows.
- A well-documented code implementation of the cost-reducing flow network that can be accessed on GitHub:

Program Construction

Python 3.7 was used to create the bifurcation finder program. Five classes of increasing complexity represented the construction of the transportation network and flow process, seen in Figure 1. The calculations for the functional G_j are made in the `Network` class. The `Flow` class iteratively finds the bifurcation point that minimizes G_j using the Nelder-Mead method [5] as implemented in `scipy.optimize.minimize` [7]. The programmatic flow process ceases when the difference between transportation network costs falls below the difference cutoff, or when the maximum number of iterations is exceeded.

Fig. 1: Classes in ascending levels of abstraction, with key attributes

IPython Notebook [6] was the framework used to create interactive visualizations accessed through the platform Binder [4]. Running the first Notebook cell brings in all needed packages and generates an ipywidget for getting program input, seen in Figure 2. The second cell performs flow minimization and outputs the results. The last cell creates an interactive plot in Bokeh [2]. Using a slider, the user can view what the transportation looks like at different steps of the flow.

Fig. 2: New networks can be created without code using this ipywidget.

Bifurcation Finder Program Examples

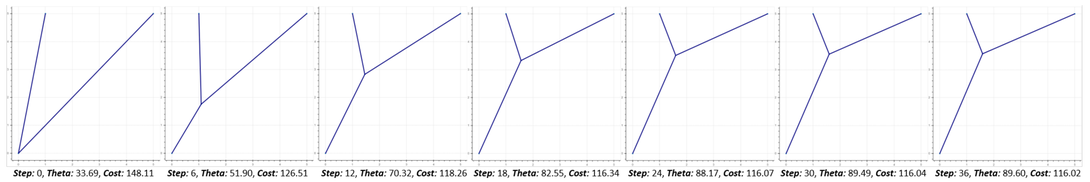

Fig. 3: Flow process on a network with non-symmetric source nodes with weight 1. An alpha value of 0.5 leads to an optimal bifurcation angle of 90 degrees [1]

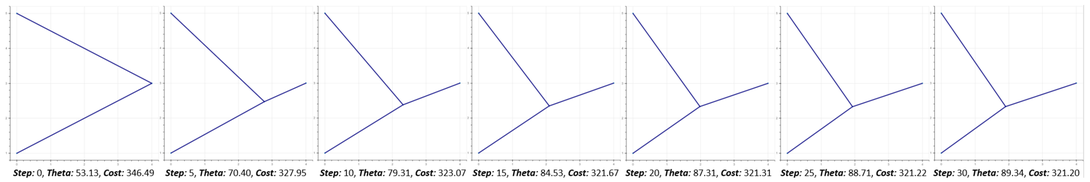

Fig. 4: Flow process with two symmetric source nodes, the top one has a weight of 1, and the bottom one has a weight of 10. Alpha is 0.5, so the optimal angle is 90 degrees.

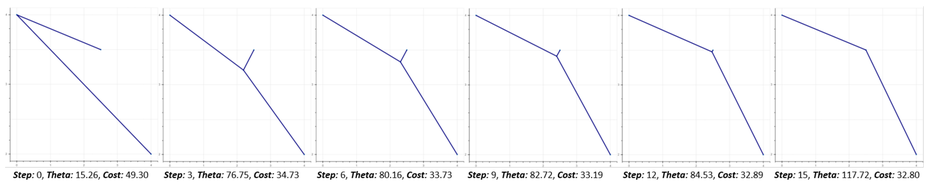

Fig. 5: Example of a network that leads to a degeneracy when alpha is 0.5. The optimal network goes from the farthest source to the nearest source, and then to the sink.

Results

The current program is capable of constructing a transportation network with a near-optimal bifurcation point given two source nodes and one sink node. The values of node coordinates and weighs must be positive. Users can easily create their own networks online, changing node values as well as parameters such as alpha.

Try the Program

Scan the QR code above or visit the GitHub repo at

`github.com/SandersKM/bifurcation-finder`

then click on the `launch binder` link in the README to try the program out yourself. For further information or the latest updates, email `sandersKM@hendrix.edu`.

Future Work

- Refine the minimization method in `scipy`
- Create further abstractions of nodes and edges
- Generalize program to work on an arbitrary number of source nodes.
- Extend the program to create transportation networks between multiple source and sink nodes.

Acknowledgements

I would like to thank Dr. Carol Ann Downes for guiding and encouraging me throughout this research project, and Dr. Brent Yorgey for providing feedback. I would also like to thank the American Mathematical Society for funding my travel to JMM.

References

[1] M. Bernot, V. Caselles, and J.M. Morel. *Optimal Transportation Networks: Models and Theory*. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 2008. ISBN: 9783540693154.

[2] Bokeh Development Team. *Bokeh: Python library for interactive visualization*. 2018.

[3] Carol Ann Downes. "A Mass Reducing Flow for Real-Valued Flat Chains with Applications to Transport Networks". PhD thesis. Rice University, 2018.

[4] Project Jupyter et al. "Binder 2.0 - Reproducible, interactive, sharable environments for science at scale". In: *Proceedings of the 17th Python in Science Conference*. Ed. by Fatih Akici et al. 2018, pp. 113–120. DOI: 10.25080/Majora-4af1f417-011.

[5] J. A. Nelder and R. Mead. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (Jan. 1965), pp. 308–313. ISSN: 0010-4620. DOI: 10.1093/comjnl/7.4.308.

[6] Fernando Pérez and Brian E. Granger. "IPython: a System for Interactive Scientific Computing". In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53.

[7] Pauli Virtanen et al. "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python". In: *arXiv e-prints*, arXiv:1907.10121 (July 2019), arXiv:1907.10121. arXiv: 1907.10121.